

Combi-nații de taste uzuale

Compilare și asamblare

Combi-nație de taste	Descriere
F7	Asamblează proiectul
Shift+F7	Compilează fișierul curent
F4	Avansează la următoarea eroare din fereastra de asamblare
F5	Execută cu depanare
Shift+F5	Execută programul

Editorul de text

Combi-nație de taste	Descriere
Ctrl+F	Caută prima apariție
F3	Caută următoarea apariție
Ctrl+F3	Caută precedenta apariție
Ctrl+A	Selectează tot
Ctrl+C	Copiază cele selectate în Clipboard
Ctrl+N	Creează un nou fișier sau proiect
Ctrl+O	Deschide un nou fișier
Ctrl+S	Salvează fișierul curent
Ctrl+V	Inserează textul aflat în Clipboard
Ctrl+W	Lansează ClassWizard
Ctrl+X	Decupează cele selectate și le plasează în Clipboard
Ctrl+Y	Repetă ultima acțiune
Ctrl+Z	Anulează ultima acțiune

Resurse

Combi-nație de taste	Descriere
Ctrl+D	Afișează ordinea de selectare dintr-o casetă de dialog
Ctrl+T	Testează controalele dintr-o casetă de dialog

Depanare

Combi-nație de taste	Descriere
F9	Stabilește sau elimină un punct de întrerupere
F10	Execută dintr-o dată linia curentă
F11	Execută linia curentă
Ctrl+B	Permite editarea punctelor de întrerupere

Jon Bates
Tim Tompkins

Utilizare Visual C++[®] 6

Traducere de Lucian Limona

Teora

Titlul original: **Using Visual C++® 6**

Traducerea din limba engleză s-a făcut după ediția originală publicată în Statele Unite ale Americii în anul 1998.

Copyright © 2001, 2000 Teora

Toate drepturile asupra versiunii în limba română aparțin Editurii **Teora**.

Reproducerea integrală sau parțială a textului sau a ilustrațiilor din această carte este posibilă numai cu acordul prealabil scris al Editurii **Teora**.

Authorized translation from the English language edition published by Que Corporation.

Copyright © 1998

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Romanian language edition published by Teora Publishing.

Copyright © 1999

Teora

Calea Moșilor nr. 211, sector 2, București

fax: 01 210.38.28

e-mail: teora@teora.kappa.ro

Teora – Cartea prin poștă

CP 79-30, cod 72450 București, România

tel: 01 252.14.31

e-mail: cpp@teora.kappa.ro

Coperta: Gheorghe Popescu

Tehnoredactare: Techno Media

Director Editorial: Diana Rotaru

Președinte: Teodor Răducanu

NOT 4666 CAL VISUAL C++6, UTILIZARE
ISBN 973-20-0306-5

Printed in Romania

Cuprinsul pe scurt

I Crearea aplicațiilor Visual C++

- 1 Proiectarea și crearea unui program în Visual C++ 20
- 2 Despre mediul de dezvoltare 35

II Casete de dialog și controale

- 3 Crearea și proiectarea casetelor de dialog 53
- 4 Utilizarea controalelor de tip buton 71
- 5 Utilizarea controalelor textuale 93
- 6 Utilizarea controalelor de tip listă 113
- 7 Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și dată/oră 136
- 8 Tratarea evenimentelor generate de mouse 168
- 9 Utilizarea controalelor ActiveX 185
- 10 Utilizarea casetelor de dialog 199

III Elementele unei aplicații

- 11 Lucrul cu imagini, bitmap-uri și pictograme 222
- 12 Utilizarea documentelor, a reprezentărilor și a cadrelor 245
- 13 Lucrul cu meniuri 269
- 14 Utilizarea barelor cu instrumente și a barelor de stare 291

IV Desenarea elementelor grafice

- 15 Despre desenarea în cadrul contextelor dispozitiv 322
- 16 Utilizarea penițelor și a pensulelor 348
- 17 Utilizarea fonturilor 378

V Tehnici avansate de lucru cu documente și vederi

- 18 Dimensionarea și derularea reprezentărilor 403
- 19 Utilizarea reprezentărilor de tip listă, arbore, editare formatată și HTML 426
- 20 Crearea de reprezentări multiple 457
- 21 Dezvoltarea aplicațiilor cu documente multiple 476
- 22 Tipărirea și previzualizarea 503

VI Transferul datelor aplicației

- 23 Salvarea, încărcarea și transferul datelor 531
- 24 Utilizarea bazelor de date și a reprezentărilor de tip înregistrare 563
- 25 Despre programarea OLE și COM 580

VII Subiecte avansate

- 26 Crearea controalelor ActiveX 604
- 27 Utilizarea depanatorului integrat 631
- 28 Utilizarea pachetelor API și SDK 654

Glosar 693

Index 706

Introducere 17

- Ce este Visual C++ 6.0 și la ce folosește? 17
- Ce este nou în această carte? 17
- Vă este utilă această carte? 18
- Convenții utilizate în această carte 18

I Crearea aplicațiilor Visual C++

1 Proiectarea și crearea unui program în Visual C++ 20

- Lansarea mediului Visual C++ 21
- Crearea unui nou proiect 21
- Selectarea tipului de proiect 22
- Denumirea proiectului și alegerea locației sale 22
- Utilizarea vrăjitorului AppWizard 23
- Utilizarea opțiunilor de bază din AppWizard 23
- Asamblarea și rularea aplicației 24
- Alegerea configurației de asamblare 24
- Efectuarea compilării și a editării de legături 25
- Rularea unei aplicații 26
- Despre interfața Windows 26
- Modificarea interfeței aplicației 26
- Adăugarea unui control buton 26
- Asocierea de cod cu interfața 30
- Testarea aplicației modificate 32
- Salvarea și închiderea proiectului 33

2 Despre mediul de dezvoltare 35

- Utilizarea mediului Developer Studio 36
- Deschiderea unui proiect existent 36
- Fereastra spațiului de lucru al proiectului 38
- Lucrul cu reprezentarea claselor 38
- Lucrul cu reprezentarea resurselor 45
- Lucrul în reprezentarea fișierelor 49
- Administrarea proiectelor 51
- Opțiunile de proiect 51
- Configurații adiționale 52

II Casete de dialog și controale

3 Crearea și proiectarea casetelor de dialog 53

- Crearea de machete pentru casetele de dialog 54
- Stabilirea valorii ID pentru caseta de dialog 58
- Utilizarea proprietăților generale ale casetei de dialog 58
- Utilizarea stilurilor pentru casete de dialog 59
- Adăugarea și poziționarea controalelor 60
- Dimensionarea controalelor 64
- Selectarea mai multor controale 65
- Alinierea controalelor 66
- Utilizarea liniilor de ghidare 66
- Organizarea controalelor din casetele de dialog 67
- Utilizarea casetelor de grupare 68
- Stabilirea ordinii de selectare 69
- Stabilirea tastelor de acces 70

4 Utilizarea controalelor de tip buton 71

- Utilizarea butoanelor de comandă 72
- Adăugarea rutinelor de tratare pentru evenimentele clic ale butoanelor 74
- Despre hărțile de mesaje 75
- Modificarea butoanelor de comandă la momentul execuției 76
- Utilizarea butoanelor de opțiune 82
- Adăugarea grupurilor de butoane de opțiune 83
- Identificarea butonului de opțiune selectat 84
- Utilizarea casetelor de validare 88
- Crearea casetelor de validare 88
- Citirea și modificarea stării casetelor de validare 89

5 Utilizarea controalelor textuale 93

- Utilizarea controalelor etichetă statică 94
- Formatarea textului din casetele de dialog 94
- Combinarea etichetelor statice și a casetelor de editare 94
- Modificarea controalelor etichetă statică la momentul execuției 95
- Utilizarea controalelor casetă de editare 100
- Adăugarea casetelor de editare 100
- Modificarea și extragerea textului din casetele de editare 102
- Tratarea mesajelor de informare asupra editării 104
- Subclasarea controalelor de editare 107
- Utilizarea controalelor de editare multi-linie 112

6 Utilizarea controalelor de tip listă 113

- Crearea controalelor de tip listă 114
- Adăugarea casetelor combinate 114
- Adăugarea controalelor arbore 116
- Adăugarea controalelor casetă cu listă 118
- Adăugarea unui control listă 119
- Adăugarea de elemente în controalele de tip listă 121
- Popularea unei casete combinate 121
- Tratarea mesajelor corespunzătoare casetelor combinate 124
- Popularea unui arbore 125
- Popularea unei casete cu listă 128
- Tratarea mesajelor corespunzătoare casetelor cu listă 130
- Popularea unui control listă 131

7 Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și dată/oră 136

- Controale orientate spre intervale de valori 137

- Utilizarea unui control indicator de evoluție 137
- Adăugarea unui control indicator de evoluție într-o casetă de dialog 137
- Maparea unei variabile peste un control indicator de evoluție 139
- Manipularea și actualizarea controlului indicator de evoluție 140
- Utilizarea unui control bară de derulare 142
- Adăugarea unui control bară de derulare într-o casetă de dialog 142
- Maparea unei variabile peste un control bară de derulare 143
- Inițializarea unui control bară de derulare 144
- Tratarea mesajelor trimise de barele de derulare 146
- Utilizarea unui control glisor 150
- Adăugarea unui control glisor într-o casetă de dialog 150
- Maparea unei variabile peste un control glisor 151
- Inițializarea unui control glisor 152
- Tratarea mesajelor trimise de controlul glisor 153
- Utilizarea controalelor de selecție a datei/orei 154
- Adăugarea unui control de selecție a datei/orei într-o casetă de dialog 155
- Maparea unei variabile peste un control de selecție a datei/orei 156
- Inițializarea unui control de selecție a datei/orei 157
- Tratarea mesajelor generate de modificarea datei 160
- Utilizarea controlului calendar 162
- Adăugarea unui control calendar într-o casetă de dialog 163

- Maparea unei variabile peste un control calendar 164
- Inițializarea unui control calendar 164
- Selecția unui interval de timp în cadrul controlului calendar 165
- Tratarea mesajelor generate de modificarea datei selectate 166

8 Tratarea evenimentelor generate de mouse 168

- Urmărirea butoanelor mouse-ului 169
- Tratarea mesajelor generate de apăsarea și eliberarea butoanelor mouse-ului 169
- Interceptarea evenimentelor dublu clic 174
- Urmărirea deplasării mouse-ului și a poziției acestuia 175
- Tratarea evenimentului de deplasare a mouse-ului 175
- Capturarea mouse-ului 179
- Implementarea unui test de clic 180
- Utilizarea clasei `CRectTracker` 181

9 Utilizarea controalelor ActiveX 185

- Selectarea și adăugarea controalelor ActiveX din galeria de componente 186
- Inspectarea controalelor ActiveX 186
- Inserarea de noi controale în proiectul curent 188
- Selectarea, dimensionarea și testarea controalelor ActiveX pornind de la paleta de controale 189
- Modificarea proprietăților controalelor în cadrul editorului de resurse 191
- Stabilirea proprietăților standard 191
- Utilizarea paginilor de proprietăți ale controalelor 191
- Utilizarea claselor asociate controalelor 193

- Adăugarea unei variabile membru de tip clasă dispecer pentru un control 193
- Citirea și scrierea prin cod a proprietăților controalelor 194
- Implementarea funcțiilor de tratare a evenimentelor ActiveX prin intermediul `ClassWizard` 196

10 Utilizarea casetelor de dialog 199

- Crearea unei casete de dialog 200
- Adăugarea unei noi resurse de tip machetă de dialog 201
- Derivarea unei clase din `CDialog` cu ajutorul `ClassWizard` 201
- Inițializarea noii clase dialog 203
- Afișarea unei casete de dialog modale 204
- Adăugarea de variabile membru pentru stocarea datelor din casetele de dialog 206
- Utilizarea schimbului și validării datelor din casetele de dialog 207
- Utilizarea funcțiilor pentru schimbul de date (DDX) 208
- Utilizarea funcțiilor pentru validarea de date (DDV) 210
- Crearea propriilor funcții de validare 212
- Utilizarea casetelor de dialog nemodale 213
- Crearea și distrugerea casetelor de dialog nemodale 213
- Înscrierea și citirea datelor dintr-o casetă de dialog nemodală 217
- Tratarea mesajului de închidere în cadrul unei casete de dialog nemodale 220
- Înlăturarea butonului de închidere 221

III Elementele unei aplicații

11 Lucrul cu imagini, bitmap-uri și pictograme 222

- Utilizarea editorului de imagini 223
- Crearea și editarea resurselor de tip pictogramă 225
- Modificarea pictogramei implicite MFC 225
- Inserarea unei noi resurse de tip pictogramă 226
- Inserarea unei noi resurse de tip bitmap 227
- Modificarea dimensiunilor și numărului de culori ale unui bitmap 228
- Importarea imaginilor 228
- Utilizarea imaginilor în casetele de dialog 230
- Stabilirea proprietăților unui control imagine 230
- Încărcarea resurselor imagine la momentul execuției 231
- Crearea butoanelor grafice 235
- Crearea de bitmap-uri pentru stările butonului 236
- Utilizarea clasei asociate butoanelor grafice 237
- Utilizarea imaginilor în cadrul controalelor 238
- Despre listele de imagini 238
- Crearea și utilizarea unei liste de imagini 240

12 Utilizarea documentelor, a reprezentărilor și a cadrelor 245

- Crearea unei aplicații SDI 246
- Despre clasele unei aplicații SDI 250
- Elementele vizuale ale unei aplicații SDI 251
- Despre machetele de document SDI 253

- Utilizarea funcțiilor oferite de infrastructura Document/Reprezentare 255
- Utilizarea împreună a documentelor și a reprezentărilor 260
- Inițializarea datelor documentului 260
- Adăugarea în document a variabilelor membru 261
- Accesarea datelor documentului din cadrul reprezentării 262
- Utilizarea resurselor standard 265
- Actualizarea reprezentării 267

13 Lucrul cu meniuri 269

- Crearea și editarea resurselor de tip meniu 270
- Adăugarea unor noi resurse de tip meniu 270
- Adăugarea titlurilor de meniu 271
- Adăugarea elementelor de meniu 271
- Atribuirea de identificatori pentru meniuri 272
- Modificarea proprietăților unui element de meniu 273
- Adăugarea de separatori 274
- Crearea elementelor de submeniu 274
- Adăugarea marcajelor de validare 274
- Specificarea tastelor de acces 275
- Tratarea comenzilor de meniu 276
- Adăugarea unei funcții de tratare pentru o comandă de meniu 276
- Adăugarea unei funcții de tratare a mesajelor provenite de la interfața cu utilizatorul 277
- Activarea și dezactivarea opțiunilor de meniu 278
- Afișarea sau înlăturarea unui marcaj de validare 279
- Modificarea dinamică a etichetelor de meniu 280
- Implementarea meniurilor de context 280

- Deschiderea unui meniu de context 281
- Tratarea comenzilor din meniul de context 284
- Crearea și accesarea obiectelor meniu 285
- Inițializarea obiectului CMenu 285
- Adăugarea dinamică a elementelor de meniu 286
- Modificarea dinamică a elementelor de meniu 289
- Înlăturarea dinamică a elementelor de meniu 290

14 Utilizarea barelor cu instrumente și a barelor de stare 291

- Personalizarea barei cu instrumente standard 292
- Despre bara cu instrumente standard 293
- Adăugarea de butoane pe bara cu instrumente prin intermediul editorului de resurse 297
- Deplasarea și înlăturarea butoanelor și adăugarea de separatori 299
- Activarea și dezactivarea butoanelor de pe bara cu instrumente 299
- Adăugarea unei bare cu instrumente proprii 300
- Adăugarea unei noi resurse de tip bară cu instrumente 300
- Adăugarea barei cu instrumente în fereastra cadru 301
- Înlăturarea și afișarea barei cu instrumente 302
- Stocarea și încărcarea pozițiilor barelor cu instrumente 303
- Utilizarea barelor de dialog 304
- Adăugarea unei resurse de tip bară de dialog 304
- Adăugarea barei de dialog în fereastra cadru 305

- Tratarea controalelor de pe bara de dialog 307
- Personalizarea barei de stare 309
- Despre bara de stare standard 310
- Adăugarea indicatorilor și a separatorilor 312
- Modificarea dinamică a dimensiunii, stilului și conținutului casetelor de indicare 315
- Despre barele suport în stil Internet Explorer 319
- Utilizarea barei suport generată de AppWizard 319
- Stabilirea unui titlu și a unui bitmap de fundal pentru bara suport 321

IV Desenarea elementelor grafice

15 Despre desenarea în cadrul contextelor dispozitiv 322

- Introducere în contextele dispozitiv 323
- Tipuri de contexte dispozitiv 323
- Utilizarea clasei CDC 324
- Utilizarea contextelor dispozitiv client 329
- Utilizarea contextelor dispozitiv de redesenare 330
- Utilizarea contextelor dispozitiv de memorie 335
- Utilizarea modurilor de mapare 337
- Moduri de mapare cu scalare liberă 340
- Determinarea caracteristicilor unui dispozitiv 342

16 Utilizarea penițelor și a pensulelor 348

- Crearea penițelor 349
- Utilizarea clasei CPen 349
- Stabilirea tipului de peniță 349
- Modificarea grosimii peniței 349
- Modificarea culorii peniței 350
- Utilizarea penițelor din stoc 351

- Selectarea penițelor în cadrul contextelor dispozitiv 352
- Ștergerea penițelor 353
- Trasarea de linii și figuri cu ajutorul penițelor 355
- Crearea unui context dispozitiv pentru desenare 355
- Deplasarea poziției peniței 355
- Desenarea de linii 357
- Desenarea prin puncte de coordonate 358
- Desenarea de cercuri și elipse 360
- Desenarea de curbe 362
- Desenarea poligoanelor 364
- Crearea pensulelor 365
- Utilizarea clasei CBrush 365
- Crearea de pensule colorate și hașurate 365
- Colorarea fundalului unei ferestre 366
- Crearea pensulelor pe bază de șabloane sau imagini 367
- Utilizarea pensulelor din stoc 369
- Selectarea pensulelor în cadrul contextelor dispozitiv 371
- Ștergerea pensulelor 372
- Desenarea de figuri umplute cu ajutorul pensulelor 372
- Desenarea de dreptunghiuri și dreptunghiuri rotunjite 372
- Desenarea de cercuri și elipse umplute 374
- Desenarea de suprafețe de coardă și rază 374
- Desenarea de poligoane 375

17 Utilizarea fonturilor 378

- Funcții pentru afișarea textului 379
- Afișarea de text simplu 379
- Stabilirea alinierii textului 380
- Modificarea culorilor de afișare și de fundal pentru text 382

- Afișarea de text opac și transparent 383
- Decuparea textului la un dreptunghi 384
- Crearea de fonturi 386
- Utilizarea clasei CFont 386
- Crearea fonturilor cu CreatePointFont() 386
- Crearea fonturilor cu CreateFont() 387
- Selectarea și alegerea fonturilor 393
- Iterarea fonturilor 393
- Utilizarea casetei de dialog pentru selecția fonturilor 397
- Afișarea de text formatat și desfășurat pe mai multe linii 400
- Ștergerea fonturilor 402

V Tehnici avansate de lucru cu documente și vederi

18 Dimensionarea și derularea reprezentărilor 403

- Tratarea redimensionării ferestrei 404
- Crearea unei aplicații exemplificative 404
- Tratarea evenimentului de dimensionare 405
- Tratarea evenimentului de fixare a dimensiunii 408
- Stabilirea de limite pentru dimensiuni 413
- Crearea unor casete de dialog dimensionabile 415
- Derularea ferestrelor 415
- Stabilirea dimensiunilor de derulare 416
- Modificarea incrementelor de derulare pentru linie și pagină 418
- Utilizarea poziției curente de derulare 420
- Tratarea mesajelor de derulare 422

19 Utilizarea reprezentărilor de tip listă, arbore, editare formatată și HTML 426

Ce sunt reprezentările de tip listă, arbore și editare formatată? 427

Crearea și utilizarea unei reprezentări de tip listă 427

Crearea unei aplicații cu reprezentare de tip listă prin intermediul AppWizard 427

Inserarea de elemente 428

Modificarea stilului unei liste 432

Adăugarea de coloane și antete asociate 434

Determinarea elementelor selectate într-o listă 438

Crearea și utilizarea unei reprezentări de tip arbore 441

Crearea unei aplicații cu reprezentare de tip arbore prin intermediul AppWizard 441

Modificarea stilului unui arbore 441

Inserarea de elemente 442

Determinarea nodului selectat 446

Controlul editării imediate 447

Crearea și utilizarea unei reprezentări de tip editare formatată 449

Crearea unei reprezentări de tip editare formatată 450

Încărcarea și salvarea textului din cadrul reprezentării 451

Formatarea paragrafelor 451

Inserarea de obiecte OLE 453

Crearea și utilizarea unei reprezentări de tip browser HTML 454

Crearea unei reprezentări HTML 454

Stabilirea adresei URL 454

Tratarea evenimentelor generate de browser 455

20 Crearea de reprezentări multiple 457

Despre reprezentările multiple 458

Utilizarea ferestrelor multi-secțiune 458

Crearea ferestrelor multi-secțiune dinamice 458

Inițializarea ferestrelor multi-secțiune dinamice 461

Crearea de ferestre multi-secțiune statice 464

Inițializarea ferestrelor multi-secțiune statice 464

Crearea unei aplicații în stilul Windows Explorer 468

Crearea dinamică a reprezentărilor multiple 469

Adăugarea și înlăturarea reprezentărilor 469

Controlul asupra creării și activării reprezentării 470

21 Dezvoltarea aplicațiilor cu documente multiple 476

Crearea unei aplicații cu interfață de tip document multiplu 477

Despre clasele unei aplicații MDI 480

Elementele vizuale ale unei aplicații MDI 481

Despre machetele de document MDI 483

Secvența de creare a documentului, reprezentării și a cadrului MDI 486

Accesarea obiectelor document/reprezentare 489

Dezvoltarea unui exemplu de aplicație MDI 490

Adăugarea de variabile membru în document 490

Accesarea datelor documentului din cadrul reprezentării 492

Modificarea datelor documentului și actualizarea reprezentării 493

Adăugarea de noi machete de document 495

22 Tipărirea și previzualizarea 503

Utilizarea funcționalității oferite de infrastructură 504

Utilizarea facilităților de tipărire implicite 504

Supradefinirea funcției OnPrint() 508

Utilizarea contextului dispozitiv de tipărire 510

Păstrarea proporției dimensiunilor 512

Paginarea și orientarea 515

Stabilirea paginilor de început și sfârșit 515

Utilizarea casetei de dialog Print 519

Utilizarea orientărilor de tip portret și peisaj 523

Crearea obiectelor GDI în OnBeginPrinting() 523

Personalizarea operației de pregătire a contextului dispozitiv 525

Suspendarea procesului de tipărire 525

Tipărirea directă, fără mijlocirea infrastructurii 526

Apelarea directă a casetei de dialog Print 526

Utilizarea funcțiilor StartDoc() și EndDoc() 528

Utilizarea funcțiilor StartPage() și EndPage() 529

VI Transferul datelor aplicației**23 Salvarea, încărcarea și transferul datelor 531**

Utilizarea serializării 532

Manipularea fișierelor în aplicațiile de tip dialog 532

Crearea obiectelor de date serializabile 533

Stocarea datelor în cadrul documentului 539

Serializarea obiectelor de date 543

Utilizarea listei cu fișierele cel mai recent utilizate 545

Înregistrarea tipurilor de documente 546

Manipularea fișierelor 546

Utilizarea clasei CFile 546

Deschiderea fișierelor 547

Citirea și scrierea unui fișier 548

Manipularea poziției curente în fișier 552

Obținerea de informații despre fișiere 553

Redenumirea și ștergerea fișierelor 555

Alte clase derivate din CFile 555

Transferul de date prin Clipboard 556

Stabilirea formatelor de date pentru Clipboard 556

Copierea datelor în Clipboard 558

Lipirea datelor din Clipboard 560

24 Utilizarea bazelor de date și a reprezentărilor de tip înregistrare 563

Utilizarea bazelor de date 564

Utilizarea bazelor de date relaționale 564

Utilizarea conectivității deschise a bazelor de date 564

Configurarea unei surse de date 566

Generarea unei aplicații cu suport pentru bazele de date 568

Includerea suportului pentru bazele de date cu ajutorul AppWizard 568

Conectarea la o bază de date 570

Interogarea bazei de date 572

Actualizarea informațiilor din baza de date 575

Asocierea dintre câmpuri și coloanele tabelor din baza de date 576

Crearea și utilizarea unei reprezentări de tip înregistrare 576

Editarea machetei reprezentării de tip înregistrare 577

Atașarea controalelor de editare la câmpurile din mulțimea de înregistrări 578

25 Despre programarea OLE și COM 580

- Programarea bazată pe componente 581
- Interfețe COM 582
- Identificatori de interfață, identificatori de clasă și identificatori globali unici 584
- Crearea de instanțe ale obiectelor COM 586
- Bibliotecile DLL de intermediere și apelurile intermediare 588
- Versiunile interfețelor 588
- Automatizarea OLE 589
- Despre interfața dispecer 589
- Utilizarea varianțelor 590
- Crearea unui server de automatizare 591
- Crearea unui client de automatizare 599
- Containere, servere și mini-servere OLE 602

VII Subiecte avansate**26 Crearea controalelor ActiveX 604**

- Crearea unei infrastructuri ActiveX prin intermediul ActiveX Control Wizard 605
- Specificarea numărului de controale, a licențierii și a suportului pentru asistență 605
- Specificarea numelor de clase și a opțiunilor de utilizare 606
- Subclasarea controalelor existente în scopul moștenirii funcționalității 607
- Utilizarea opțiunilor avansate pentru controlul ActiveX 608
- Implementarea controlului 608
- Desenarea controlului 609
- Tratarea acțiunilor utilizatorului și a evenimentelor generate 612
- Un rapid test parțial al controlului 613

- Implementarea generării de evenimente 614
- Crearea interfeței pentru proprietăți 617
- Implementarea proprietăților generice 617
- Adăugarea unei pagini pentru proprietățile generice de culoare 619
- Adăugarea proprietăților specifice 619
- Adăugarea într-o pagină de proprietăți a controalelor corespunzătoare proprietăților specifice 622
- Conservarea proprietăților 624
- Compilarea și înregistrarea controlului 626
- Diferitele fișiere sursă 627
- Crearea fișierelor pentru biblioteca de tipuri și pentru licențiere 627
- Înregistrarea controlului 628
- Testarea cu ajutorul containerului de test pentru controale ActiveX 628
- Selectarea și inserarea controlului 628
- Testarea proprietăților controlului 629
- Testarea proprietăților ambientale 629
- Jurnalizarea evenimentelor generate 630

27 Utilizarea depanatorului integrat 631

- Crearea informațiilor pentru depanare și inspectare 632
- Utilizarea configurațiilor pentru depanare și pentru versiunea finală 632
- Stabilirea opțiunilor și a nivelelor pentru depanare 633
- Crearea și utilizarea informațiilor pentru inspectare 635
- Utilizarea depanării la distanță și a depanării imediate 638
- Diagnosticarea și execuția controlată 639
- Utilizarea macrodefiniției TRACE 639

- Utilizarea macrodefinițiilor ASSERT și VERIFY 642
- Utilizarea punctelor de întrerupere și a execuției controlate a programului 644
- Utilizarea facilității de editare și continuare 647
- Urmărirea variabilelor din program 647
- Alte ferestre pentru depanare 648
- Alte instrumente pentru depanare 650
- Utilizarea programului Spy++ 650
- Process Viewer 652
- OLE/COM Object Viewer 652
- MFC Tracer 653

28 Utilizarea pachetelor API și SDK 654

- Introducere în API și SDK 655
- Implementarea de sunet și grafică rapide cu DirectX 655
- Utilizarea interfeței DirectSound 656
- Utilizarea interfeței DirectDraw 664
- Utilizarea interfeței Direct3D 673

- Utilizarea interfeței DirectPlay 674
- Utilizarea interfeței DirectInput 674
- Utilizarea interfeței DirectSetup 674
- Crearea de mesaje și poșta electronică prin intermediul MAPI 674
- Utilizarea interfeței MAPI de bază 675
- Utilizarea AppWizard pentru adăugarea facilității de expediere a mesajelor prin intermediul MAPI 677
- Utilizarea bibliotecilor multimedia video și audio 681
- Utilizarea interfeței pentru controlul multimedia 681
- Mesajele de notificare MCI 685
- Adăugarea unei ferestre MCI 689

Glosar 693**Index 706**

Despre autori

Jon Bates a lucrat în ultimii 15 ani la un număr mare de proiecte software din domeniile comercial, industrial și militar. În prezent activează pe cont propriu, ca și consultant de proiect și autor de programe, fiind specializat în dezvoltarea aplicațiilor Visual C++ pentru Windows NT/95 și 98.

Jonathan și-a început cariera scriind jocuri pentru micro-calculatoare răspândite la acel moment și a lucrat pe mai multe sisteme de operare, printre care CPM, DOS, TRIPOS, UNIX și Windows, și cu mai multe limbaje de asamblare, de a treia generație și orientate pe obiect.

A creat programe de sistem și aplicații diverse, cum ar fi drivere, poștă electronică, modelarea producției, editare video, analiza imaginilor, rețele și telecomunicații, achiziție de date, sisteme de control, previziuni și costuri sau programe de vizualizare. De asemenea, a scris mai multe articole în revistele de specialitate privind o serie de subiecte.

Jonathan locuiește împreună cu soția sa, Ruth, și cu câinele său, Chaos, în centrul răcoroasei Britanii. Atunci când nu se ocupă de calculator, lui îi place să adoarmă și să viseze fractali.

Puteți să-l contactați pe Jonathan la adresa jon@chaos1.demon.co.uk și să-i vizitați situl de Web la adresa www.chaos1.demon.co.uk.

Tim Tompkins este în prezent directorul pentru dezvoltare de software în cadrul unui producător european specializat în sisteme informaționale de conducere integrate. El răspunde de proiectarea și implementarea aplicațiilor comerciale scrise în Visual C++.

Cariera de programator a lui Tim a început în 1986 cu scrierea de aplicații de analiză statistică în COBOL, pe sisteme mainframe IBM. De atunci a acumulat opt ani de experiență în C și Visual C++ și a lucrat pe o multitudine de sisteme de operare, printre care CPM, DOS, UNIX, XENIX, Windows NT/95 și 98.

A proiectat și implementat un număr mare de module de program, planificarea producției, colectarea datelor de vânzări, controlul acțiunilor și facturare, previziuni și costuri și pontare.

Tim locuiește împreună cu soția sa, Tracey, și cu calul acesteia (care nu intră în casă), Oliver, pe ținuturile verzi și plăcute ale Angliei. Atunci când are timp, se urcă la volanul mașinii sale decapotabile (chiar și pe ploaie).

Puteți să-l contactați pe Tim la adresa tinytim@globalnet.co.uk.

Calculatoarele personale pe care rulează una din variantele sistemului de operare Windows au devenit un peisaj întâlnit în case și birouri de pretutindeni. Popularizarea Internetului și a disponibilităților multimedia ale PC-urilor au accelerat creșterea cererii de aplicații tot mai diverse și mai puternice. Pare la modă stigmatizarea lui Microsoft exclusiv datorită succesului său. Și totuși, succesul Microsoft și atenția permanentă pe care firma o acordă standardizării au permis programatorilor talentați sub Windows să-și valorifice capacitatea din ce în ce mai bine pe piața muncii. Cererea pentru un astfel de talent și, implicit, răsplata acordată, fie aceasta financiară sau de altă natură, nu pot decât să crească pe măsură ce societatea acceptă și integrează calculatoarele tot mai mult în modul de viață al fiecărui individ.

Ce este Visual C++ 6.0 și la ce folosește?

Visual C++ 6.0 este ultima și cea mai bună versiune a compilatorului Microsoft Visual C/C++. Produsul a devenit mult, mult mai mult decât un simplu compilator. Sunt incluse clasele fundamentale Microsoft (Microsoft Foundation Classes), care simplifică și accelerează dezvoltarea aplicațiilor Windows. Sunt incluse editoare sofisticate de resurse în scopul proiectării casetelor de dialog complexe, a meniurilor, barelor cu instrumente, imaginilor și a multor alte elemente ce compun aplicațiile Windows moderne. Este oferit un excelent mediu de dezvoltare integrat, numit Developer Studio, care prezintă forme grafice ale structurii aplicației pe măsură ce o dezvoltați. Un instrument pentru depanare integrat perfect vă permite să inspectați în detaliu fiecare aspect din cadrul unui program aflat în execuție. Iar acestea sunt doar câteva din numărul copleșitor de facilități oferite de Visual C++ 6.0, care vă ajută să dezvoltați aplicații rapide și complete folosind cele mai recente tehnologii din Windows.

Ce este nou în această carte?

Visual C++ 6.0 aduce o mulțime de noi facilități, iar această carte încearcă să cuprindă cât mai multe dintre acestea.

Noile controale, așa cum este controlul de selecție a datei pe care se poate să-l fi întâlnit în aplicația de poștă electronică Microsoft Outlook, sunt acum disponibile pentru utilizare în propriile aplicații (despre aceasta vom vorbi în capitolul 7, „Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și dată/oră”).

Elementelor din casetele combinate le pot fi asociate acum imagini, iar acestea sunt afișate în caseta de selecție și în lista derulantă ale noului control extins casetă combinată (prezentat în capitolul 11, „Lucrul cu imagini, bitmap-uri și pictograme”).

Acele bare cu instrumente plate pe care le-ați văzut în Office 97 și în Internet Explorer 4 sunt acum integrate în cadrul claselor fundamentale, putând fi utilizate în propriile dumneavoastră aplicații (vom vorbi despre ele în capitolul 15, „Utilizarea barelor cu instrumente și a barelor de stare”).

Puteți acum să apelați Internet Explorer din cadrul ferestrelor aplicației pentru a vizualiza pagini de Web și elemente HTML (așa cum veți vedea în capitolul 20, „Utilizarea reprezentărilor de tip listă, arbore, text formatat și HTML”).

Puternicul AppWizard poate să creeze acum și mai multe tipuri de structuri inițiale de aplicații, astfel că noua dumneavoastră aplicație poate avea întreaga funcționalitate a unor aplicații cu forme multiple de vizualizare, așa cum este Windows Explorer, fără a scrie nici o singură linie de cod (subiect discutat în capitolul 21, „Crearea de reprezentări multiple”).

Programarea bazată pe componente și modelul componentelor obiectuale distribuite (DCOM) reprezintă domenii noi cu o creștere semnificativă în dezvoltarea modernă a aplicațiilor (prezentate în capitolul 26, „Despre OLE și programarea COM”).

Cele mai recente versiuni de interfețe API și pachete SDK, așa cum este DirectX Game SDK, sunt prezentate în capitolul 32, „Lucrul cu API și SDK”.

Toate acestea și multe alte facilități mai mărunte ale noii versiuni Visual C++ 6.0 sunt prezentate în cuprinsul acestei cărți.

Vă este utilă această carte?

Cartea de față presupune că ați acumulat o anumită experiență în ceea ce privește programarea C++. Nu este necesară experiență în Windows sau în utilizarea claselor fundamentale Microsoft, dar orice cunoștințe dobândite vă vor ajuta în mod evident. Este bine să fiți familiar într-o oarecare măsură cu utilizarea sistemelor de operare Windows 95/98 sau NT și a interfeței grafice cu utilizatorul a acestora.

Puteți fie să folosiți această carte pentru a vă clădi de la temelie o sumă de cunoștințe privind programarea aplicațiilor Windows în C++, parcurgând capitolele unul după celălalt, sau puteți să consultați diverse capitole pentru a studia aspecte care vă interesează.

Tim și cu mine am avut experiența cărților care par a nu avea acel plus de concret necesar într-o aplicație reală. Așa că am făcut un efort pentru a acoperi acest ultim kilometru, explicând și prezentând secvențe de cod care reflectă problemele și necesitățile de care ne-am lovit în aplicațiile comerciale la care am lucrat. Înarmat cu această carte, veți putea să scrieți aplicații complexe și competitive cu ajutorul unuia dintre cele mai flexibile și mai confortabile medii de dezvoltare existente: Visual C++ 6.0.

Convenții utilizate în această carte

Pentru a spori claritatea informațiilor prezentate în această carte, au fost utilizate următoarele convenții de scriere:

- Cuvintele sau expresiile folosite pentru prima dată apar cu caractere *cursive* și sunt explicate în glosar. Cuvintele afișate pe ecran sunt tipărite cu caractere **aldine**. Tastele de acces și de prescurtare sunt prezentate cu o **subliniere** care arată tasta în cauză.
- Tot ceea ce are legătură cu codul și numele claselor folosite în cod apar cu caractere monotipate.
- Liniile de cod care sunt prea lungi pentru lățimea unei pagini sunt tipărite pe două linii, asociate prin intermediul unui simbol de continuitate a codului (↪). Astfel de linii trebuie privite ca o singură linie de cod continuă din cadrul programului.

Vă urăm succes și sperăm că veți scrie cu plăcere aplicații în Visual C++ 6.0 cu ajutorul acestei cărți.

CAPITOLUL

1

Proiectarea și crearea unui program în Visual C++

Generarea unei aplicații Windows prin intermediul AppWizard

Modificarea aspectului unei aplicații prin intermediul editorului de resurse

Scrierea de cod pentru personalizarea comportamentului unei aplicații

Multe persoane care nu au programat niciodată o aplicație Windows consideră această activitate ca fiind una dificilă. Dar un instrument potrivit poate întotdeauna să înlesnească o activitate, iar pentru dezvoltarea de programe cu interfață grafică nu veți găsi un instrument mai bun decât Visual C++ 6. De fapt, prin intermediul Visual C++ 6 puteți să creați și să rulați o aplicație Windows în mai puțin de un minut, însă noi vom porni ceva mai încetșor.

Lansarea mediului Visual C++

Pentru a lansa Visual C++, selectați **Microsoft Visual C++ 6.0** din cadrul meniului **Start** de pe bara de sarcini. Odată mediul lansat, veți vedea fereastra Microsoft Visual Studio.

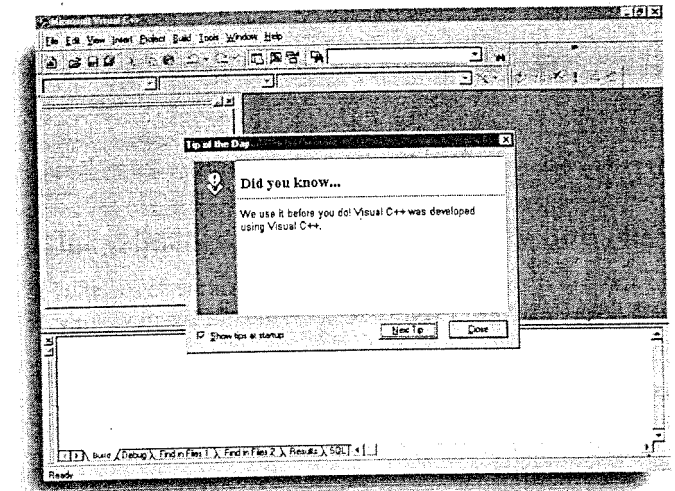
Pseudonim Visual Studio

Visual Studio a purtat numele de Developer Studio. Este posibil să întâlniți și acronimul IDE (Integrated Development Environment – mediu integrat de dezvoltare), folosit uneori ca și referire la Visual Studio.

Fereastra Visual Studio este afișată la lansarea mediului Visual C++. Visual Studio este numele dat interfeței cu utilizatorul a lui Visual C++, interfață afișată în figura 1.1. Visual Studio reprezintă suprafața dumneavoastră de lucru. Nu vă îngrijorați dacă fereastra Visual Studio nu vă apare identică cu cea prezentată în diverse figuri. Pe parcursul cărții veți învăța să regăsiți opțiunile necesare.

FIGURA 1.1

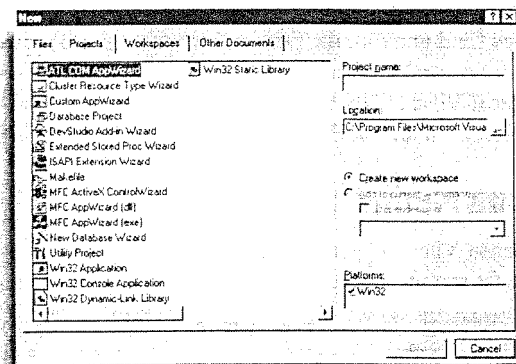
Fereastra Visual Studio.



Crearea unui nou proiect

Pentru a începe o nouă aplicație trebuie să creați mai întâi un proiect. Un proiect este folosit pentru administrarea tuturor elementelor care compun un program Visual C++ și care au ca rezultat o aplicație Windows. Selectați comanda **New** a meniului **File** pentru a crea un nou proiect. Caseta de dialog New va fi afișată ca în figura 1.2.

FIGURA 1.2
Caseta de dialog New.



Selectarea tipului de proiect

Mai întâi trebuie să specificați tipul proiectului pe care doriți să-l creați. Efectuați un clic pe pagina **Projects** din caseta de dialog New, în cazul în care această pagină nu este deja selectată. Aici este afișată o listă cu toate tipurile de proiecte pe care le puteți crea. Pentru exemplul din acest capitol, selectați **MFC AppWizard (exe)** din lista cu tipurile de proiect. Selectarea acestei opțiuni înseamnă că proiectul va avea ca rezultat un program executabil Windows standard.

De unde vine MFC?

Visual C++ este însoțit de biblioteca de clase de bază Microsoft (Foundation Classes). Biblioteca MFC reprezintă o mulțime de clase C++ predefinite. AppWizard generează un schelet de aplicație prin crearea unor clase derivate din clasele aflate în biblioteca MFC. Utilizarea claselor de bază este ilustrată pe parcursul întregii cărți.

Denumirea proiectului și alegerea locației sale

Orice proiect are nevoie de un nume. Acest nume se specifică în caseta **Project Name** din caseta de dialog New. Pentru exemplul nostru, tastați **Minute** în caseta **Project Name** din cadrul casetei de dialog New.

Caseta Location este folosită pentru precizarea directorului în care vor fi plasate fișierele proiectului. În cazul exemplului de față nu este nevoie să modificați această locație.

Denumirea proiectelor

Puteți să atribuiți proiectului un alt nume decât cel pe care vi-l sugerăm. Pentru a evita confuzii, însă, vă recomandăm să folosiți numele sugerate în carte la crearea exemplelor prezentate. În ceea ce privește propriile aplicații, alegeți numele care reflectă scopul programului. Pentru acest prim program vă propunem **Minute**, deoarece la începutul capitolului ne-am lăudat că o aplicație nouă poate fi creată în mai puțin de un minut.

Calea afișată inițial în caseta **Location** depinde de opțiunile exprimate la instalarea lui Visual C++. Pentru a modifica această locație, fie editați calea explicit, fie efectuați un clic

pe butonul aflat în partea dreaptă a casetei **Location**. Locația implicită se bazează pe numele proiectului – în exemplul nostru, **C:\Program Files\Microsoft Visual Studio\MyProjects\Minute**.

Utilizarea vrăjitorului AppWizard

După stabilirea opțiunilor din cadrul casetei de dialog New, efectuați un clic pe **OK** pentru a iniția generarea proiectului. Lăsați-l pe AppWizard să umească lucrurile pentru dumneavoastră. Scopul său în viață este să creeze un schelet de program pe care să-l puteți extinde ulterior. Acest lucru este realizat permițându-vă să specificați tipul de program dorit, după care este folosită *biblioteca MFC* pentru generarea fișierelor care vor forma împreună un proiect Visual Studio.

VEDEȚI...

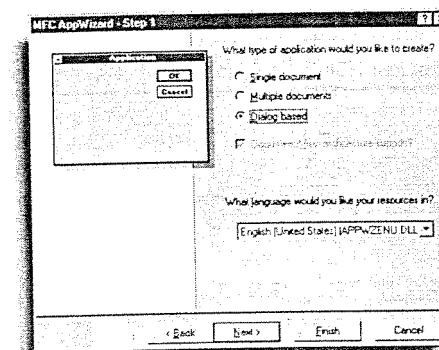
► Pentru mai multe detalii privind crearea unei aplicații de tip dialog, vedeți pagina 54.

Utilizarea opțiunilor de bază din AppWizard

Caseta de dialog AppWizard (ilustrată în figura 1.3) vă oferă trei opțiuni pentru tipul de interfață a aplicației. În cazul proiectului **Minute** veți utiliza o interfață de tip dialog. Selectați butonul de opțiune **Dialog Based**. Puteți, de asemenea, să selectați limba care va fi folosită pentru resurse. Nu este necesar să modificați conținutul casetei combinate **Language** în cazul programului **Minute**.

FIGURA 1.3

Caseta de dialog MFC AppWizard – Step1 permite selectarea tipului de interfață.

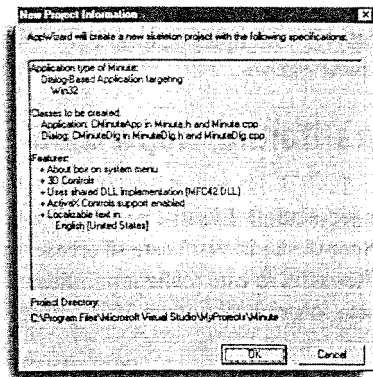


Acum ați precizat toate informațiile necesare pentru ca AppWizard să poată crea proiectul. Efectuați un clic pe butonul **Finish**. Va fi afișată caseta de dialog **New Project Information**, ilustrată în figura 1.4.

AppWizard vă prezintă această casetă de dialog pentru a confirma detaliile proiectului pe care este gata să-l creeze. Puteți vedea aici numele claselor C++ din proiect și numele fișierelor care vor fi create. În lista **Features** puteți să vedeți, de asemenea, funcționalitatea asigurată de AppWizard. Efectuați un clic pe **OK** în cadrul casetei de dialog **New Project Information**.

FIGURA 1.4

Caseta de dialog New Project Information.



AppWizard și-a îndeplinit sarcina și în Visual Studio este deschis acum proiectul nou creat – Minute. În acest moment, deși nici măcar nu ați văzut vreo linie de cod sursă C++, aveți la dispoziție o aplicație Windows completă și perfect funcțională.

Secțiunea afișată de către Visual Studio în partea stângă se numește *secțiunea spațiului de lucru*. Remarcați că, după ce proiectul a fost creat, secțiunea spațiului de lucru oferă trei pagini: **ClassView**, **ResourceView** și **FileView**. Aceste pagini vă permit accesarea oricărei componente a proiectului. Puteți să modificați dimensiunea secțiunii spațiului de lucru și a altor secțiuni care apar în Visual Studio prin efectuarea unui clic pe marginea secțiunii și deplasarea mouse-ului în timp ce țineți butonul apăsat.

Reluarea opțiunilor AppWizard

AppWizard este utilizat exclusiv pentru crearea noulor proiecte. Nu puteți reveni la casetele de dialog cu opțiuni AppWizard în cadrul unui proiect existent. Dacă descoperiți că ați făcut o alegere greșită și doriți să reluați etapele AppWizard, trebuie să înlăturați mai întâi proiectul existent. Pentru a înlătura un proiect, ștergeți directorul acestuia. Spre exemplu, pentru a relua crearea proiectului Minute, apelați la Explorer pentru a șterge directorul C:\Program Files\Microsoft Visual Studio\MyProjects\Minute.

Asamblarea și rularea aplicației

Procesul de asamblare are ca rezultat fișierul executabil Windows corespunzător proiectului. În cazul proiectului Minute, acest fișier va fi numit Minute.exe. Odată generat acest fișier, îl puteți rula din Visual Studio.

Alegerea configurației de asamblare

Puteți determina Visual Studio să creeze fie o *versiune pentru depanare*, fie o *versiune finală* a unui fișier executabil. Aici este vorba despre *configurația de asamblare*. În mod implicit este asamblată o versiune pentru depanare; folosiți această opțiune implicită în toate exemplele dacă nu se propune explicit o altă variantă. Pentru programul Minute nu este necesară modificarea configurației. În mod implicit, Visual Studio va genera o aplicație care conține informații pentru depanare. Aceste informații vă permit să inspectați

codul executat și să verificați valorile variabilelor. Inserarea informațiilor pentru depanare duce, însă, la o creștere a dimensiunii fișierului executabil și la o scădere a performanțelor. Configurația pentru versiunea finală assemblează executabilul fără informații pentru depanare, fiind folosită de regulă atunci când aplicația este livrată către un client.

Executabilele Visual C++ sunt fișiere Windows EXE standard

Fișierul Minute.exe este similar cu orice alt fișier executabil Windows. Prin urmare, el poate fi executat din Explorer, sau puteți să creați o scurtătură la acesta și să o plasați pe suprafața de lucru.

VEDEȚI...

➤ Pentru a afla mai multe despre depanarea proiectelor, vedeți pagina 631.

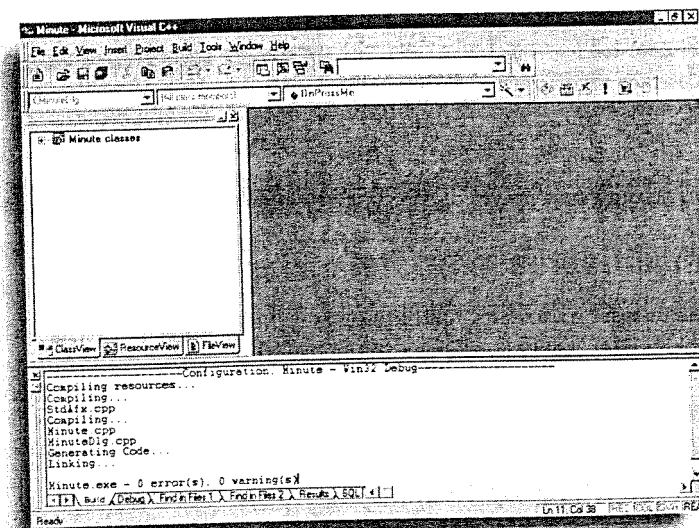
Efectuarea compilării și a editării de legături

Procesul de asamblare efectuează compilarea fișierelor C++ individuale dintr-un proiect, după care rezultatele sunt legate împreună pentru a forma fișierul executabil. Pentru a asambla proiectul Minute, efectuați un clic pe butonul **Build**  sau selectați **Build Minute.exe** din meniul **Build** sau apăsați tasta F7.

Minute.exe se află în subdirectorul /Debug din directorul /Minute al proiectului. Acest subdirector conține, de asemenea, fișierele obiect ale programelor din cadrul proiectului. Subdirectorul /Debug a fost creat din cauză că ați ales o configurație pentru depanare, în timp ce o configurație pentru versiunea finală ar fi plasat fișierele într-un subdirector numit /Release. Pagina **Build** a secțiunii Output înfățișează informații privind procesul de asamblare. În cazul în care codul sursă conține erori, acestea sunt afișate în cadrul paginii **Build**, ilustrată în figura 1.5. Deoarece AppWizard a generat întreg codul, nu ar trebui să apară erori.

FIGURA 1.5

Asamblarea proiectului Minute.

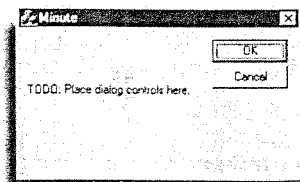


Rularea unei aplicații

Pentru a rula aplicația, efectuați un clic pe butonul **Execute** sau selectați **Execute Minute.exe** din meniul **Build** sau folosiți combinația de taste Ctrl+F5. Fereastra principală a aplicației va apărea pe ecran ca în figura 1.6.

FIGURA 1.6

Aplicația Minute.



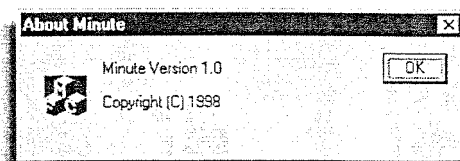
Despre interfața Windows

Felicitări! Ați creat o aplicație Windows în C++. Veți vedea că aceasta oferă deja unele elemente standard de interfață care apar practic în toate programele Windows. Fereastra aplicației conține două butoane, **OK** și **Cancel**, și afișează un text. Există, totodată, o bară de titlu care afișează o pictogramă asociată, numele aplicației și un buton de închidere. Bara de titlu conține și un meniu de sistem și poate fi utilizată pentru a deplasa fereastra pe ecran prin efectuarea unui clic asupra ei și menținerea butonului mouse-ului apăsat.

Efectuați un clic pe imaginea MFC din colțul stânga sus al ferestrei Minute. Selectați **About Minute** din meniul de sistem care apare. Veți vedea că programul creat dispune chiar de o a doua casetă de dialog, titrată About Minute, care afișează informații despre aplicație, așa cum se vede în figura 1.7.

FIGURA 1.7

Casetă de dialog About a aplicației Minute.



Modificarea interfeței aplicației

Elementele vizuale ale unui proiect se numesc *resurse*. Spre exemplu, casetele de dialog, pictogramele și meniurile constituie resurse. Numit sugestiv editor de resurse, acesta este instrumentul din Visual Studio care se folosește pentru proiectarea resurselor de diferite tipuri și modificarea aspectului efectiv al aplicației.

Adăugarea unui control buton

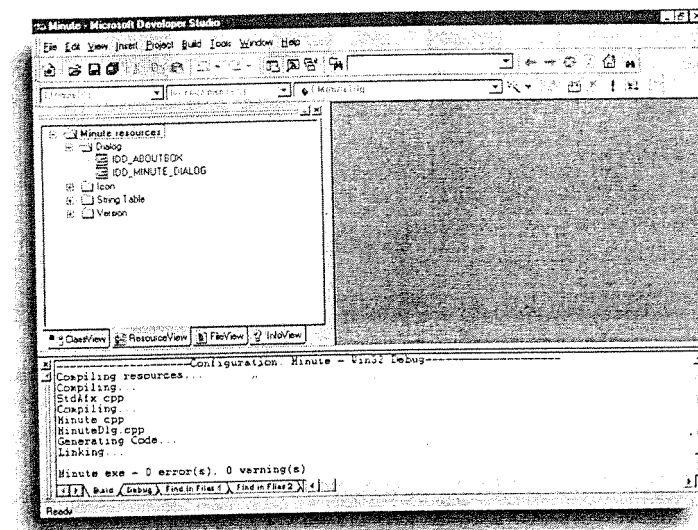
Înainte de a adăuga un nou buton, va trebui să deschideți macheta casetei de dialog.

Deschiderea casetei de dialog Minute

1. Selectați pagina **ResourceView** a secțiunii spațiului de lucru al proiectului. Va fi afișată lista cu resursele proiectului.
2. Expandați lista de resurse prin efectuarea unui clic pe semnul + din stânga etichetei **Minute Resources** și expandați catalogul **Dialog**. Așa cum se vede în figura 1.8, apar doi identificatori de dialog, **IDD_ABOUTBOX** și **IDD_MINUTE_DIALOG**.

FIGURA 1.8

Identificatorii de dialog din pagina ResourceView.



3. Efectuați un dublu clic pe identificatorul **IDD_MINUTE_DIALOG**. În consecință va fi afișată macheta ferestrei dialog principale a aplicației Minute, ca în figura 1.9. La rularea programului, dialogul va apărea așa cum apare în editorul de resurse. Acum puteți modifica macheta dialogului prin intermediul editorului de resurse.

În continuare trebuie să adăugați un nou buton în cadrul machetei de dialog Minute.

Adăugarea unui buton în cadrul dialogului Minute

1. Înainte de a adăuga butonul, înlăturați controlul etichetă **TODO** care apare în centul machetei dialogului Minute. Efectuați un clic pe textul **TODO: place dialog controls here;** în jurul textului este afișat un dreptunghi de formatare (vedeți figura 1.10). Apăsați tasta Delete. Controlul etichetă este înlăturat de pe macheta dialogului.
2. Selectați controlul buton din caseta cu controale, așa cum se vede în figura 1.11.
3. Deplasați mouse-ul deasupra machetei dialogului. Aflat pe suprafața machetei, cursorul mouse-ului devine o cruce pentru a indica poziția în care va fi plasat noul buton. Plasați crucea departe de butoanele **OK** și **Cancel** și apoi efectuați un clic cu mouse-ul. Va fi afișat un nou buton, etichetat **Button1**, ilustrat în figura 1.12.

FIGURA 1.9
Resursa de tip dialog
IDD_MINUTE_DIALOG.

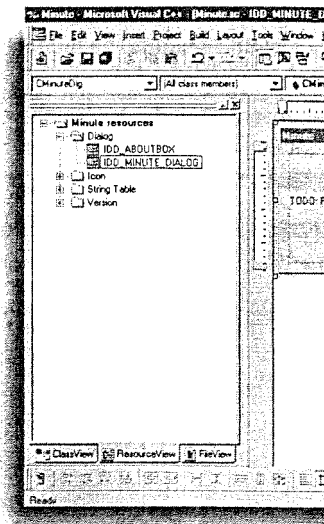
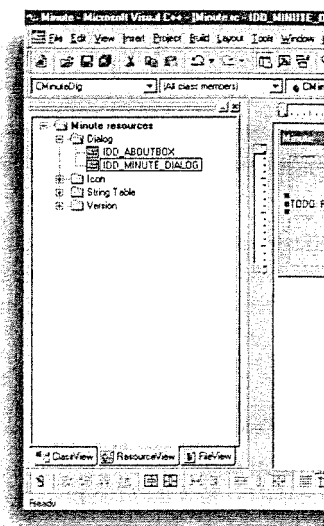


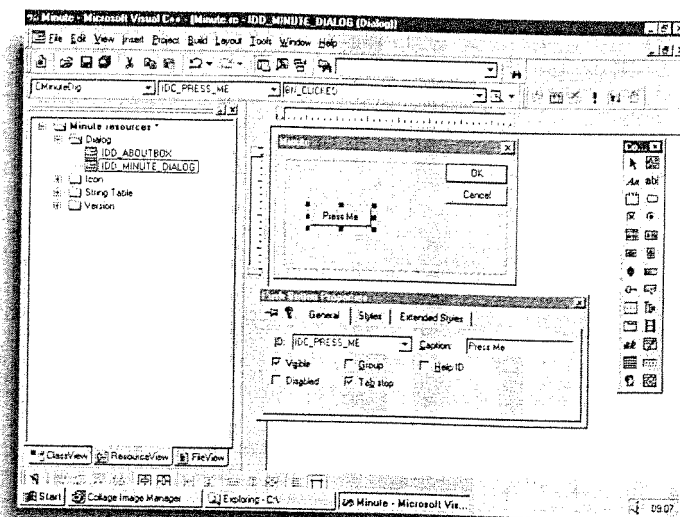
FIGURA 1.10
Înlăturarea textului implicit
din cadrul casetei de dialog.



4. Pentru a modifica eticheta butonului, asigurați-vă că este încadrat de un dreptunghi de formatare, ca în figură selectat, selectați-l prin efectuarea unui clic pe suprafața sa. Este afișată caseta de dialog Push Button.
5. Tastați IDC_PRESS_ME în cadrul casetei combinate Button Properties, înlocuind textul implicit IDC_BUTTON1. Textul IDC_PRESS_ME are o semnificație mai evidentă.

FIGURA 1.13

Modificarea etichetei și a identificatorului pentru noul buton.



Asocierea de cod cu interfața

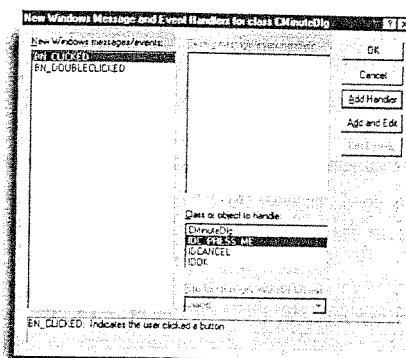
Acum aveți un nou buton care de-abia așteaptă să fie apăsat, așa că veți vrea să reacționați cumva. În acest scop va trebui să scrieți cod, însă nu trebuie să vă agitați prea mult din cauză că nu veți scrie personal decât o singură linie.

Asocierea de cod cu un buton

1. Deschideți dialogul Minute în cadrul editorului de resurse. În acest scop vedeți pașii prezentați în secțiunea anterioară, „Adăugarea unui control buton”.
2. Efectuați un clic cu butonul drept pe butonul **Press Me** de pe suprafața dialogului, după care selectați opțiunea **Events** din meniul contextual afișat. Va fi deschisă caseta de dialog **New Windows Message and Event Handlers** corespunzătoare clasei dialog, așa cum se vede în figura 1.14.

FIGURA 1.14

Adăugarea de cod pentru a răspunde la evenimentul de efectuare a unui clic asupra butonului.

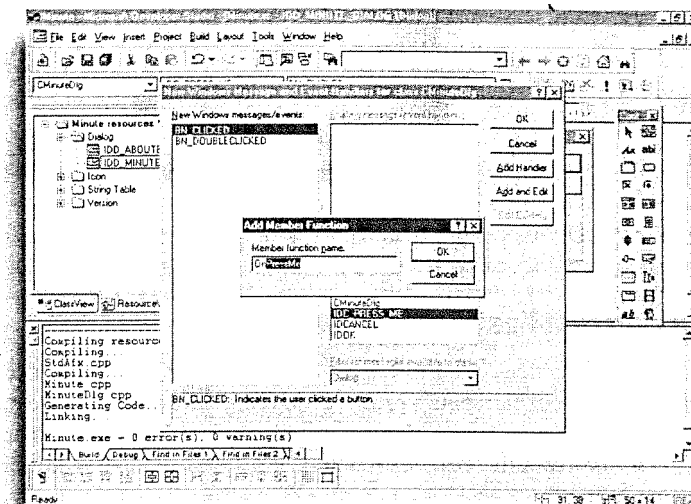


La efectuarea unui clic asupra butonului **Press Me** (în timpul rulării aplicației), Windows trimite dialogului Minute un mesaj. Caseta de dialog **New Windows Message and Event Handlers** vă permite să interceptați acest mesaj și să executați o anumită secvență de cod care să răspundă la eveniment.

3. Selectați **BN_CLICKED** din lista **New Windows Messages/Events**. Observați că primul element din listă este selectat în mod implicit.
4. Efectuați un clic pe butonul **Add and Edit**. Va fi afișată caseta de dialog **Add Member Function**, ilustrată în figura 1.15. Aici veți da un nume funcției din program care va fi apelată de fiecare dată când caseta de dialog primește mesajul **BN_CLICKED** pentru butonul **Press Me**.

FIGURA 1.15

Denumirea funcției care va trata evenimentul.



5. Efectuați un clic pe **OK** pentru a accepta numele implicit **OnPressMe**. Corpul noii funcției apare în fereastra editorului, ca în figura 1.16.

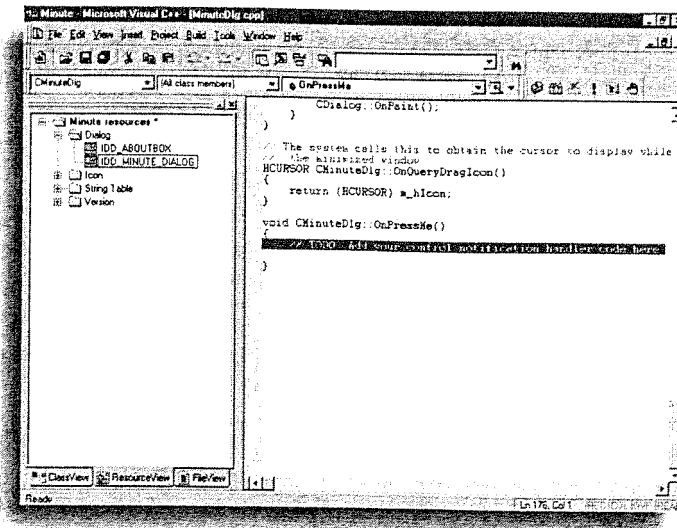
Funcția este adăugată ca și membru al clasei **CMinuteDlg**. Această clasă a fost creată și denumită în mod automat la crearea proiectului de către AppWizard. În acest moment funcția nu face nimic, așa că va trebui să adăugați cod.

6. Editați funcția **OnPressMe** astfel încât să fie identică cu secvența din listingul 1.1.

LISTINGUL 1.1 LST01_1.CPP – Implementarea răspunsului la efectuarea unui clic asupra unui buton

```
1 void CMinuteDlg::OnPressMe()
2 {
3     // TODO: Add your control notification handler code
4     MessageBox("Thanks, I needed that!");
5 }
```

FIGURA 1.16
Scheletul funcției membru
OnPressMe().



Prin folosirea funcției predefinite `MessageBox()`, oricând se efectuează un clic pe butonul **Press Me** va fi afișat într-o fereastră mică mesajul `Thanks, I needed that!`.

Testarea aplicației modificate

Atunci când aduceți modificări unui proiect și doriți să vedeți dacă acestea funcționează corect, va trebui să repetați procesul de asamblare și rulare. Efectuați un clic pe butonul **Build** sau selectați **Build Minute.exe** din meniul **Build** sau apăsați **F7**. Astfel se va relua asamblarea fișierului executabil, fiind incluse toate modificările aduse resurselor sau codului.

Modificările sunt salvate automat la asamblarea proiectului

Visual Studio salvează automat orice fișier modificat înainte de a începe asamblarea proiectului, astfel că nu mai trebuie să vă amintiți să salvați fiecare fișier editat.

Dacă în timpul compilării apar erori, acestea vor fi afișate în pagina **Build** a secțiunii **Output**, așa cum se vede în partea inferioară a ferestrei Visual Studio ilustrată în figura 1.17. În cazul în care au apărut erori, asigurați-vă că ați tastat linia de cod din cadrul funcției `OnPressMe()` exact ca în listingul 1.1. Tastați încă o dată linia de cod dacă este necesar și efectuați un clic pe butonul **Build** până când nu mai apar erori.

Efectuați un clic pe butonul **Execute** sau selectați **Execute Minute.exe** din meniul **Build** sau apăsați **Ctrl+F5**. Dialogul **Minute** ar trebui să fie afișat ca în figura 1.18.

Efectuați un clic pe butonul **Press Me**. Ca urmare ar trebui să apară o fereastră de mesaj, similară cu cea din figura 1.19, care afișează mesajul `Thanks, I needed that!`.

Efectuați un clic pe **OK** în cadrul casetei de mesaj. Dacă efectuați un nou clic asupra butonului **Press Me**, veți primi același mesaj. Aceasta se întâmplă deoarece la fiecare clic asupra butonului programul accesează aceeași funcție `OnPressMe()`.

FIGURA 1.17
Erorile sunt afișate în pagina
Build.

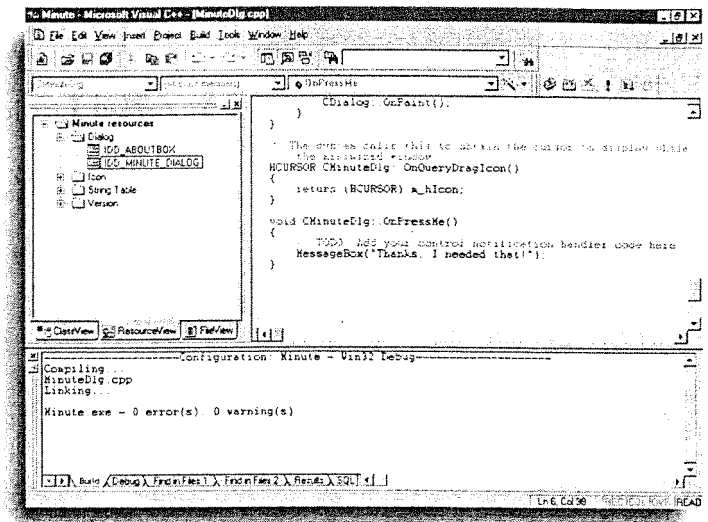


FIGURA 1.18
Rularea aplicației Minute
după modificarea interfeței.

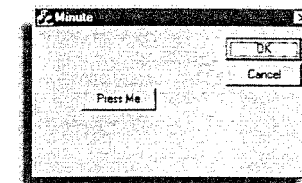
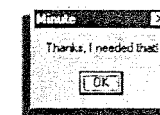


FIGURA 1.19
Testarea răspunsului
corespunzător din partea
codului.



Salvarea și închiderea proiectului

În lucrul cu Visual Studio vă veți obișnui să treceți de la editarea fișierelor sursă la modificarea resurselor și la o mie de alte operații. Devine astfel o binecuvântare faptul că procesul de asamblare salvează automat modificările efectuate înainte de a realiza asamblarea propriu-zisă. Puteți, însă, să salvați un fișier anume în orice moment prin efectuarea unui clic pe butonul **Save**. Este posibil, în plus, să salvați toate fișierele modificate prin efectuarea unui clic pe butonul **Save All**.

Indicatorul de modificare *

Atunci când există modificări care nu au fost salvate, numele fișierului respectiv apare în bara de titlu din Visual Studio însoțit de un asterisc. Asteriscul apare și alături de dosarul Resource din pagina **ResourceView** atunci când modificări efectuate asupra resurselor nu au fost salvate.

Salvarea unui proiect nu necesită nici o operație. Odată ce ați terminat lucrul cu un proiect, este suficient să-l închideți. Pentru a închide un proiect selectați **Close Workspace** din meniul **File** sau pur și simplu închideți Visual Studio.

CAPITOLUL**2****Despre mediul de dezvoltare**

Familiarizarea cu mediul Developer Studio

Navigarea în cadrul ferestrei Project Workspace și printre elementele acesteia

Despre opțiunile și configurațiile legate de administrarea proiectelor

Utilizarea mediului Developer Studio

Mediul Developer Studio utilizat de Visual C++ pare destul de complex la prima vedere. O numărătoare rapidă arată că există peste 100 de opțiuni de meniu și aproximativ tot atâtea butoane de pe bara cu instrumente care pot fi selectate. Multe dintre acestea conduc la casete de dialog complexe și la pagini de proprietăți care conțin numeroase opțiuni. Funcționalitatea atât de bogată oferită de Developer Studio este justificată de faptul că acest mediu este utilizat pe scară largă pentru a produce aplicații complexe, profesionale. Nu vă lăsați intimidat. Pentru a deveni eficient nu trebuie să învățați decât o parte din întreaga funcționalitate.

Personalizarea mediului Developer Studio

Există multe modalități prin care mediul Developer Studio poate fi adaptat la necesitățile dumneavoastră. Opțiunile de meniu și barele cu instrumente pot fi personalizate prin selectarea opțiunii **Customize** din meniul **Tools**. Puteți personaliza, de asemenea, fonturile și culorile din ferestrele de editare, precum și alte elemente, prin intermediul comenzii **Options** din meniul **Tools**.

Deschiderea unui proiect existent

În cazul în care încă nu ați făcut-o, lansați Visual C++ prin intermediul meniului **Start** de pe suprafața de lucru. În capitolul 1, „Proiectarea și crearea unui program în Visual C++”, ați creat un proiect intitulat **Minute**. Acum veți deschide acel proiect și-l veți utiliza pentru a efectua un tur al mediului Developer Studio.

Cea mai rapidă modalitate de a redeschide un proiect este să apăsați la lista **Recent Workspaces**. În acest scop, efectuați un clic pe meniul **File**; apoi selectați proiectul dorit din cadrul listei **Recent Workspaces**. Rețineți că această listă conține un număr limitat de proiecte.

Pentru a deschide un proiect care nu figurează în lista **Recent Workspaces**, selectați **Open Workspace** din meniul **File**. Pe ecran este afișată fereastra **Open Workspace**, înfățișată în figura 2.1.

Accesați directorul folosit pentru proiectul **Minute** (în cazul în care ați optat pentru directorul implicit, locația proiectului ar trebui să fie **C:\Program Files\Microsoft Visual Studio\MyProjects\Minute**). Selectați din listă fișierul **Minute.dsw** și efectuați un clic pe **Open**.

Indiferent de metoda utilizată, proiectul ar trebui să fie acum deschis, cu titlul său afișat în bara de titlu a ferestrei Developer Studio. Mediul de lucru a reținut unde ați rămas și a deschis automat ultimul fișier sursă editat (vedeți figura 2.2). Pentru a înlătura fereastra **Output** care apare în partea inferioară a ecranului, efectuați un clic pe cruciulița din colțul stânga sus al ferestrei (vedeți poziția cursorului de mouse din figura 2.2).

Încărcarea ultimului proiect odată cu lansarea

Ultimul proiect la care ați lucrat poate fi deschis automat de fiecare dată când lansați Developer Studio. În acest scop, selectați **Tools, Options**. Apoi selectați pagina **Workspace** și validați opțiunea **Reload Last Workspace On Startup**.

FIGURA 2.1

Deschiderea unui proiect existent prin selectarea fișierului **.dsw**.

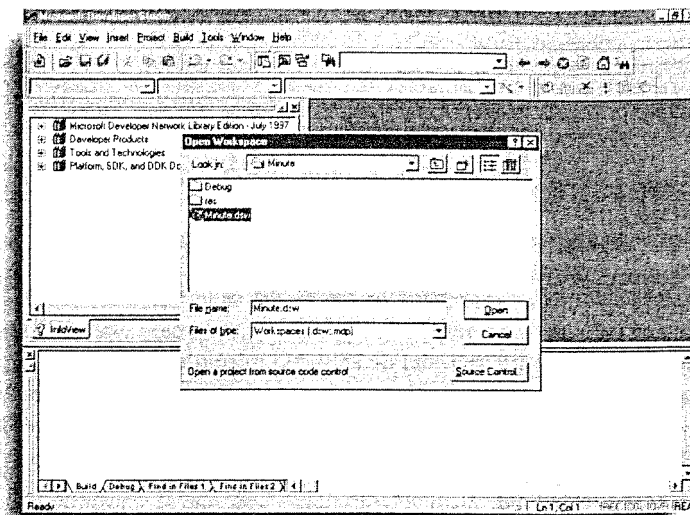
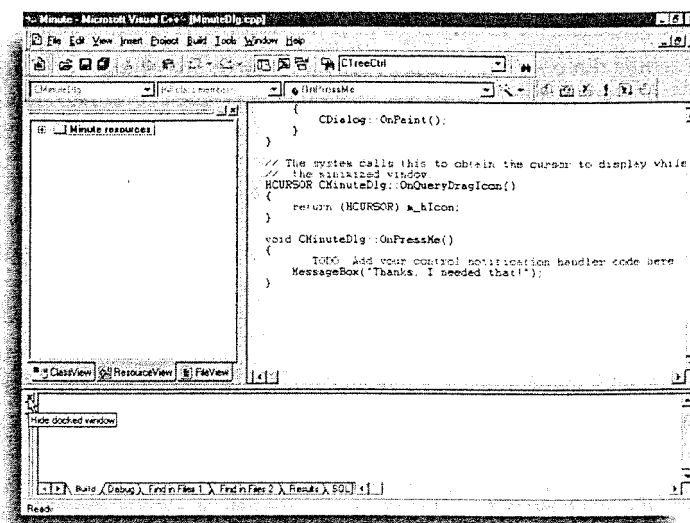


FIGURA 2.2

Developer Studio deschide un proiect în starea în care acesta a fost închis ultima dată.



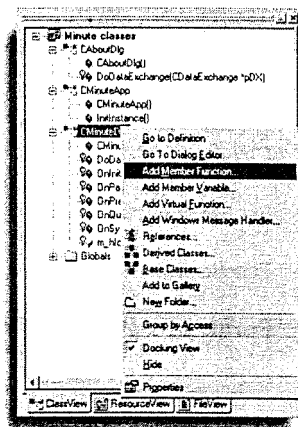
Poate vă întrebați care este semnificația termenului *spațiu de lucru* și prin ce diferă acesta de un proiect. Un spațiu de lucru poate să conțină mai multe proiecte. În toate exemplele din această carte, fiecare spațiu de lucru va conține un singur proiect, dar la dezvoltarea unor aplicații mai întinse devine adeseori de preferat administrarea unificată a mai multor proiecte.

unui element. Pentru a afișa meniul contextual pentru un element din arborele de clase, selectați respectivul element cu butonul drept al mouse-ului. Elementul în cauză va fi evidențiat și alături de el va fi afișat meniul de context. Meniul contextual afișat depinde de tipul elementului selectat.

Efectuați un clic pe numele clasei `CMinuteDlg` pentru a afișa meniul contextual al clasei, înfățișat în figura 2.4.

FIGURA 2.4

Selectarea unui element din reprezentarea claselor duce la afișarea unui meniu contextual.

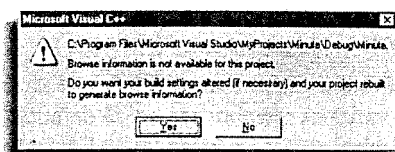


Meniul contextual vă permite să efectuați diverse operații asupra clasei selectate. Selectarea opțiunii **Go to Definition** are același efect cu efectuarea unui dublu clic pe numele clasei, și anume accesarea fișierului sursă care conține definiția clasei. Patru dintre opțiunile meniului sunt folosite pentru adăugarea de funcții sau variabile în cadrul clasei. Veți utiliza imediat o parte din acestea, dar mai întâi voi explica pe scurt alte opțiuni din meniul contextual.

Efectuați un clic pe opțiunea **References** și pe ecran va apărea mesajul din figura 2.5.

FIGURA 2.5

Un fișier de navigare este necesar pentru unele din opțiunile meniului.



Efectuați un clic pe **Yes** pentru a genera fișierul de navigare. Proiectul va fi asamblat automat. Odată încheiat procesul, închideți fereastra Output prin efectuarea unui clic pe cruciulița aflată în colțul din stânga sus.

Informațiile de navigare aduc facilități suplimentare de navigare, disponibile prin intermediul meniului contextual din ClassView.

Opțiunea **References** are ca efect afișarea tuturor locațiilor din proiect care fac referire la elementul selectat. Pentru fiecare referință sunt afișate numele fișierului sursă și numărul

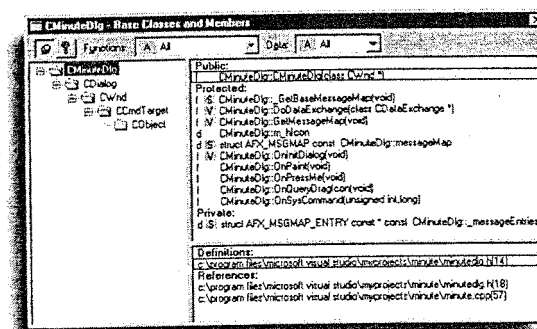
liniei, fiindu-vă oferită posibilitatea de a accesa direct oricare dintre linii. Opțiunea **Derived Classes** afișează detalii privind clasa selectată și clasele derivate din aceasta. Opțiunea **Base Classes** afișează, după cum probabil ați intuit, detalii privind clasa selectată și toate clasele sale de bază, inclusiv clasele din biblioteca MFC (vedeți figura 2.6).

Atunci când apeleți meniul contextual pentru funcțiile sau variabilele membru, opțiunea **Calls** afișează toate funcțiile apelate de către funcția membru selectată, iar opțiunea **Called By** afișează toate locațiile din proiect care conțin un apel al funcției.

Opțiunea **Group By Access** din meniul contextual este folosită pentru a ordona membrii clasei. Aceștia sunt afișați în ordine alfabetică în cazul în care opțiunea **Group By Access** nu este bifată; în caz contrar, membrii sunt afișați în ordinea tipului de acces – privat, protejat și apoi public.

FIGURA 2.6

Selectarea opțiunii **Base Classes** are ca efect afișarea ierarhiei de clase.



Acum veți aduce modificări clasei `CMinuteDlg`. Mai întâi, asigurați-vă că clasa `CMinuteDlg` este selectată și apeleți meniul contextual. Vă amintiți că meniul contextual este afișat prin efectuarea unui clic cu butonul drept. Selectați **Add Member Variable** din meniul de context. Va fi afișată caseta de dialog **Add Member Variable**, ilustrată în figura 2.7. Așa cum o sugerează și numele său, această casetă de dialog este folosită pentru a adăuga o variabilă în cadrul clasei. Va trebui să specificați tipul, numele și modul de acces pentru noua variabilă.

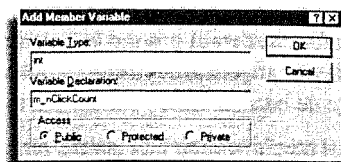
Adăugarea de membri al clasei fără editarea codului sursă

Casetele de dialog **Add Member Variable** și **Add Member Function**, accesibile prin intermediul meniului contextual din Class View, adaugă automat liniile de cod necesare în fișierele antet și de implementare corespunzătoare clasei selectate.

Tastați **int** în caseta **Variable Type**. Tastați `m_nClickCount` în caseta **Variable Declaration**. Lăsați modul de acces să fie **Public**. Astfel creați în cadrul clasei `CMinuteDlg` o variabilă de tip întreg numită `m_nClickCount`. Această variabilă va fi utilizată pentru a reține de câte ori s-a efectuat clic asupra butonului **Press Me**. Odată ce ați introdus informațiile necesare, efectuați un clic pe **OK**.

FIGURA 2.7

Adăugarea unei noi variabile
membru prin intermediul
ClassView.



Noua variabilă ar trebui să fie acum afișată în pagina ClassView ca element subordonat clasei CMinuteDlg. Efectuați un dublu clic pe elementul m_nClickCount. Va fi deschisă fereastra editorului și cursorul este plasat în linia de cod care a fost adăugată pentru definirea noii variabile.

Deoarece această variabilă va reține de câte ori s-a efectuat clic asupra butonului **Press Me**, ea va trebui să aibă valoarea inițială zero. Locul potrivit pentru inițializarea variabilei este constructorul clasei CMinuteDlg. Pentru a adăuga codul de inițializare, efectuați un clic pe elementul corespunzător funcției membru constructor CMinuteDlg (acesta este primul element subordonat clasei CMinuteDlg). Fereastra editorului va conține fișierul sursă corespunzător. Pentru inițializarea variabilei contor, adăugați linia de cod 10 din listingul 2.1.

LISTINGUL 2.1 LST02_1.CPP – Inițializarea unei variabile membru în constructorul clasei

```
1 CMinuteDlg::CMinuteDlg(CWnd* pParent /*=NULL*/)
2 : CDialog(CMinuteDlg::IDD, pParent)
3 {
4     ///{AFX_DATA_INIT(CMinuteDlg)
5         // NOTE: the ClassWizard will add member
6         // initialization here
7     ///}AFX_DATA_INIT
8     // Note that LoadIcon does not require a subsequent
9     // DestroyIco in Win32
10    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
11    m_nClickCount = 0; — 1
12 }
```

1 Variabila m_nClickCount este inițializată cu zero în constructorul clasei casetă de dialog.

Acum numărătoarea începe de la zero; mai departe va trebui să incrementăm contorul la fiecare clic asupra butonului **Press Me**. Efectuați un dublu clic pe funcția membru OnPressMe() a clasei CMinuteDlg. În fereastra editorului veți vedea funcția pe care ați adăugat-o în exemplul din primul capitol. Vă amintiți că această funcție este apelată atunci când caseta de dialog primește mesajul BN_CLICKED corespunzător butonului IDC_PRESS_ME.

Pentru a incrementa variabila contor, adăugați linia 5 din codul prezentat în listingul 2.2.

LISTINGUL 2.2 LST01_2.CPP – Numărarea clicurilor efectuate asupra butonului Press Me

```
1 void CMinuteDlg::OnPressMe() — 1
2 {
3     MessageBox("Thanks, I needed that!");
4
5     m_nClickCount++; — 2
6 }
```

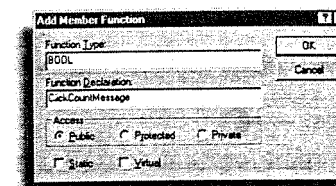
1 OnPressMe() este o funcție pentru tratarea mesajelor care este apelată de fiecare dată când se efectuează un clic pe butonul IDC_PRESS_ME.

2 Pe lângă afișarea unui mesaj pentru utilizator, funcția incrementează o variabilă contor pentru a număra de câte ori a fost apăsat butonul.

Variabila contor reține acum numărul de clicuri. Următorul pas este să adăugați o nouă funcție membru. Efectuați un clic cu butonul drept pe numele clasei CMinuteDlg pentru a apela meniul contextual. Selectați **Add Member Function**. Va fi afișată caseta de dialog Add Member Function (vedeți figura 2.8). Această casetă de dialog este folosită pentru adăugarea unei funcții membru într-o clasă.

FIGURA 2.8

Adăugarea unei noi funcții
membru prin intermediul
meniului contextual din
ClassView.



Specificați tipul funcției (acesta este tipul rezultatului întors de către funcție). Tastați **BOOL** în caseta **Function Type**. În acest mod indicăm că funcția va întoarce una din două stări posibile, și anume **TRUE** sau **FALSE**.

Trebuie să stabilim un nume pentru funcție. Tastați **ClickCountMessage** în caseta **Function Declaration**. Dacă noua funcție necesită parametri, aceștia pot fi introduși de asemenea în caseta **Function Declaration**.

Atenție: adăugarea de membri nu este supusă validării

Nu are loc nici un fel de validare atunci când adăugați membri ai claselor prin intermediul casetelor de dialog Add Member. Spre exemplu, este posibil să creați o funcție de tip **BOO** care va produce erori de compilare. Înainte de a efectua clic pe **OK** asigurați-vă de corectitudinea sintaxei.

Ar trebui ca butonul de opțiune care stabilește modul de acces **Public** să fie selectat, iar casetele de validare **Static** și **Virtual** trebuie lăsate ambele invalidate. Efectuați un clic pe **OK**.

Noua funcție ar trebui să apară acum în secțiunea ClassView ca și element subordonat clasei CMinuteDlg. Corpul acestei funcții este vizibil în fereastra editorului. Adăugați liniile de cod 3 – 18 din listingul 2.3.

LISTINGUL 2.3 LST02_3.CPP – Mesajul afișat depinde de numărul de clicuri efectuate

```
1 BOOL CMinuteDlg::ClickCountMessage()
2 {
3     if (m_nClickCount == 0)
4     {
5         MessageBox("You haven't clicked Press Me", "Don't Leave",
6             MB_ICONSTOP); — 1
7         return FALSE; — 2
8     }
9     if (m_nClickCount > 1)
10    {
11        CString str;
12        str.Format("Press Me was clicked %d times", m_nClickCount); — 3
13        MessageBox(str);
14    }
15    else
16    {
17        MessageBox("Press Me was clicked once");
18    }
19    return TRUE; — 4
20 }
```

- 1 Funcția `MessageBox()` are mai multe forme. Aici este transmis indicatorul `MB_ICONSTOP`, ceea ce duce la afișarea unui semn de stop alături de textul mesajului.
- 2 Se întoarce `FALSE` deoarece variabila contor are valoarea zero.
- 3 Funcția `Format()` a clasei `CString` este utilizată pentru a compune un mesaj textual care este transmis apoi funcției `MessageBox()`.
- 4 Se întoarce `TRUE` în cazul în care variabila contor este mai mare sau egală cu unu.

Ceea ce facem este să testăm valoarea `m_nClickCount`, afișând un mesaj corespunzător.

În cazul primului apel al funcției `MessageBox()` din linia 5 sunt transmiși doi parametri adiționali. Primul dintre aceștia, `Don't Leave`, reprezintă titlul casetei de mesaj. `MB_ICONSTOP` reprezintă un indicator care are ca efect afișarea unei pictograme cu semnul stop.

În linia 10, variabila `str` este de tipul clasei `CString`, aceasta fiind una dintre mai multe clase utile oferite de MFC. `CString` implementează multe operații uzuale de programare legate de manipularea șirurilor de caractere. Dacă programarea în C vă este familiară, funcția `Format()` (folosită în linia 11) pe care o oferă `CString` se comportă similar cu

funcția `printf()` din C. În fine, observați că funcția întoarce `FALSE` (în linia 6) în cazul în care contorul este zero și `TRUE` (în linia 18) pentru orice altă valoare.

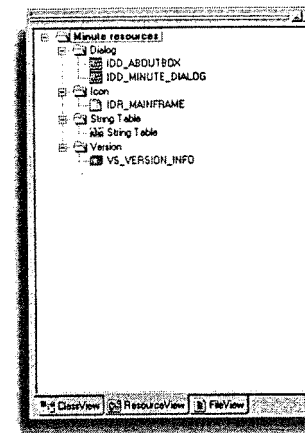
Noua noastră funcție este gata, însă nu este folosită încăieri în program. Despre acest lucru ne vom ocupa în secțiunea următoare.

Lucrul cu reprezentarea resurselor

Efectuați un clic pe eticheta de pagină **ResourceView** de la baza ferestrei spațiului de lucru al proiectului. Va fi afișată o listă cu resursele din cadrul proiectului (vedeți figura 2.9).

FIGURA 2.9

Selectarea paginii **ResourceView** vă permite să manipulați resursele din cadrul proiectului.



Mai devreme ați folosit sumar reprezentarea resurselor în scopul creării unui buton. Iată o descriere a diverselor alte tipuri de resurse.

Această reprezentare afișează toate resursele folosite în proiect. Noțiunea de *resursă* cuprinde elementele vizuale ale unui proiect. De pildă, casetele de dialog sunt resurse, la fel ca și meniurile sau pictogramele.

Separarea resurselor de cod

Motivul pentru care resurse de genul meniurilor, al casetelor de dialog și al tabelelor de șiruri sunt separate de codul sursă este acela că se dorește posibilitatea de modificare a acestora independent de acesta. Astfel se înlesnește foarte mult dezvoltarea de aplicații internaționale. Spre exemplu, toate șirurile dintr-o tabelă de șiruri pot fi traduse într-o a doua limbă, fără a fi necesară modificarea codului sursă.

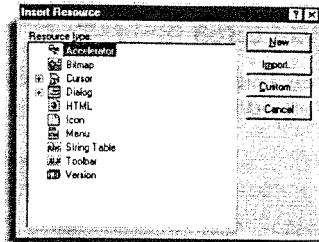
Asemeni reprezentării claselor, reprezentarea resurselor afișează un meniu contextual la efectuarea unui clic cu butonul drept asupra unui element. Selectați **Insert** din meniul contextual **Minute Resources**. Pe ecran va apărea caseta de dialog **Insert Resource**, ca în figura 2.10. Aici puteți vedea diferitele tipuri de resurse care pot fi utilizate într-un proiect Visual C++.

O tabelă de acceleratori reprezintă o listă de asocieri între combinații de taste și comenzi din program. O denumire uzuală pentru accelerator este cea de combinație de taste. O

combinație de taste reprezintă o modalitate mai rapidă de a executa o comandă decât selectarea acesteia dintr-un meniu. Spre exemplu, la scrierea acestei cărți am utilizat de multe ori Ctrl+X și am găsit această soluție mult mai convenabilă decât selectarea comenzii **Cuț** din meniul **Edit**.

FIGURA 2.10

Selectarea opțiunii **Insert** a meniului contextual din cadrul reprezentării resurselor afișează tipurile posibile de resurse.



O resursă de tip *bitmap* este o imagine de o dimensiune oarecare compusă din puncte. Fiecare punct poate avea o culoare distinctă. Bitmap-urile sunt utilizate pentru afișarea imaginilor pe care le vedeți pe barele cu instrumente, iar ceea ce vedeți pe ecranul introductiv afișat la lansarea Visual C++ este o imagine bitmap mai mare.

Un *cursor* este o resursă de tip imagine. Cursele sunt folosite în principal ca reprezentări ale indicatorului mouse-ului (cursorul standard din Windows este o săgeată). Trăsătura distinctivă a unui cursor este prezența unui punct senzitiv, care este folosit pentru a urmări poziția acestuia pe ecran.

O *pictogramă* este, de asemenea, o resursă de tip imagine. Proiectul Minute conține o pictogramă care se numește `IDR_MAINFRAME`. Efectuați un dublu clic pe elementul `IDR_MAINFRAME` și în fereastra editorului va fi deschisă pictograma MFC standard.

O *resursă de tip dialog* este o machetă de fereastră care poate conține celelalte tipuri de resurse, așa cum sunt meniurile și butoanele. Editarea resurselor de tip dialog este discutată în capitolul 3, „Crearea și proiectarea casetelor de dialog”.

O *resursă de tip meniu* conține exact ceea ce sugerează numele său, și anume un meniu. De regulă, o aplicație include un meniu în partea superioară a ferestrei sale principale (așa cum este și cazul cu Developer Studio), însă pot exista mai multe meniuri. De exemplu, este posibilă adăugarea într-un proiect a unei resurse de tip meniu și utilizarea acesteia ca și meniu de context.

O *resursă de tip tabelă de șiruri* ar trebui să conțină toate șirurile de caractere folosite într-o aplicație. Fiecare element are asociat un identificator unic, utilizat pentru a referi șirul respectiv în cadrul codului sursă (vedeți figura 2.11).

Motivul pentru care toate șirurile sunt introduse într-o tabelă de șiruri și nu direct în fișierele sursă este înlesnirea traducerii în alte limbi. Dacă toate textele utilizate într-o aplicație se află într-o tabelă de șiruri, este posibilă crearea unei a doua tabelă într-o altă limbă. După lansarea aplicației este utilizată tabela de șiruri corespunzătoare.

FIGURA 2.11

Un exemplu de tabelă de șiruri.

ID	Value	Caption
ID_EDIT_CLEAR	57633	Erase the selected text
ID_EDIT_CLEAR_ALL	57634	Erase everything in the text area
ID_EDIT_COPY	57635	Copy the selection and put it on the Clipboard
ID_EDIT_CUT	57636	Cut the selection and put it on the Clipboard
ID_EDIT_FIND	57637	Find the specified text in the text area
ID_EDIT_PASTE	57638	Paste the text from the Clipboard into the text area
ID_EDIT_REPEAT	57639	Repeat the last action in the text area
ID_EDIT_REPLACE	57640	Replace the selected text with the text in the text area
ID_EDIT_SELECT_ALL	57641	Select the entire document
ID_EDIT_UNDO	57642	Undo the last action in the text area
ID_EDIT_UNDO_REPEAT	57643	Repeat the previously undone action in the text area
ID_WINDOW_SPLIT	57644	Split the active window into panes
ID_APP_ABOUT	57645	Display program information, version number and copyright
ID_APP_EXIT	57646	Quit the application, prompts to save documents if needed
ID_NEXT_PANE	57647	Switch to the next window pane
ID_PREV_PANE	57648	Switch back to the previous window pane
ID_INDICATOR_EXT	57649	EXT
ID_INDICATOR_CAPS	57650	CAP
ID_INDICATOR_NUM	57651	NUM
ID_INDICATOR_SCROLL	57652	SCRL
ID_INDICATOR_OVR	57653	OVR
ID_INDICATOR_REC	57654	REC

O *resursă de tip bară cu instrumente* conține o mulțime de butoane. De regulă, fiecare buton reprezintă o comandă de meniu. Pe suprafața butoanelor pot fi desenate imagini sub formă de bitmap-uri și fiecare buton poate avea asociată o explicație care va fi afișată atunci când mouse-ul se deplasează deasupra butonului.

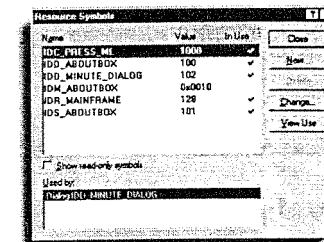
O *resursă de tip versiune* conține informații despre aplicații, ca de pildă firma producătoare, numele produsului, un număr de versiune și așa mai departe. Aceste informații se află într-un format standard care permite accesarea lor din alte aplicații.

Resurse noi pot fi adăugate în orice moment. La crearea, fiecare resursă primește automat un nume unic. De exemplu, o nouă machetă de dialog ar putea primi numele `IDD_DIALOG1`. Programatorii înlocuiesc de obicei numele generate automat cu nume de resurse mai sugestive pentru contextul programului, așa cum am făcut și noi atunci când am înlocuit numele `IDC_BUTTON1` cu `IDC_PRESS_ME`. Pentru a afișa numele tuturor resurselor din proiect, selectați **Resource Symbols** din meniul **View**. Va fi afișată caseta de dialog Resource Symbols (vedeți figura 2.12).

Puteți utiliza caseta de dialog Resource Symbols pentru a identifica o resursă anume pe baza numelui său simbolic prin selectarea numelui în cadrul listei și efectuarea unui clic pe butonul **View Use**.

FIGURA 2.12

Selectarea opțiunii **Resource Symbols** are ca efect afișarea tuturor resurselor din proiect.



Fiecare resursă din cadrul proiectului poate fi editată prin efectuarea unui dublu clic pe numele acesteia din secțiunea ResourceView. Instrumentele și opțiunile disponibile în fereastra editorului se modifică în concordanță cu tipul resursei editate. Despre crearea și editarea diferitelor tipuri de resurse vom discuta în capitole următoare, la momentul oportun.

Acum vom definiția exemplul la care ați lucrat. Prin intermediul secțiunii ResourceView, urmați pașii de mai jos.

Supradefinirea comportamentului pentru butoanele din cadrul dialogului

1. Efectuați un dublu clic pe elementul `IDD_MINUTE_DIALOG` pentru a deschide macheta dialogului.
2. Efectuați un dublu clic pe butonul **OK**.
3. Este afișată caseta de dialog **Add Member Function**. Efectuați un clic pe **OK** pentru a accepta numele implicit al funcției membru: `OnOK`. Scheletul noii funcției este acum creat și afișat în fereastra editorului, gata pentru adăugarea de cod.
4. Editați funcția implicită `OnOK()` în concordanță cu listingul 2.4.
5. Efectuați un dublu clic pe elementul `IDD_MINUTE_DIALOG` pentru a deschide din nou macheta dialogului.
6. Efectuați un dublu clic pe butonul **Cancel**.
7. Efectuați un clic pe **OK** pentru a accepta numele implicit al funcției membru: `OnCancel`.
8. Editați funcția implicită `OnCancel()` în concordanță cu listingul 2.5.

LISTINGUL 2.4 LST02_4.CPP – Supradefinirea funcției implicite `OnOK`

```
1 void CMinuteDlg::OnOK() — ❶
2 {
3     if ( ClickCountMessage() == TRUE )
4     {
5         CDialog::OnOK(); — ❷
6     }
7 }
```

❶ Această funcție supradefinește implementarea implicită `CDialog::OnOK()`, care reprezintă funcția de tratare a mesajelor pentru butonul **OK**.

❷ Este apelată implementarea funcției `OnOK()` din clasa de bază, ceea ce va duce la închiderea casetei de dialog.

LISTINGUL 2.5 LST02_5.CPP – Supradefinirea funcției implicite `OnCancel()`

```
1 void CMinuteDlg::OnCancel() — ❶
2 {
3     if ( ClickCountMessage() == TRUE )
4     {
5         CDialog::OnCancel(); — ❷
6     }
7 }
```

❶ Această funcție supradefinește implementarea implicită `CDialog::OnCancel()`, care reprezintă funcția de tratare a mesajelor pentru butonul **Cancel**.

❷ Este apelată implementarea funcției `OnCancel()` din clasa de bază, ceea ce va duce la închiderea casetei de dialog.

Ați adăugat două funcții. Prima dintre acestea supradefinește funcția implicită `CDialog::OnOK()`, iar cea de a doua supradefinește funcția `CDialog::OnCancel()`. Ambele funcții au ca efect închiderea casetei de dialog, dar acum sunt apelate numai în cazul în care funcția `ClickCountMessage()` întoarce valoarea `TRUE`. Dacă nu sunteți lămurit în legătură cu ce ați făcut, asamblați și rulați proiectul.

Fără a efectua un clic pe butonul **Press Me**, încercați să efectuați clic pe **OK** sau pe **Cancel**. Ar trebui să fie afișat mesajul ilustrat în figura 2.13, iar aplicația nu se mai încheie, ci își continuă execuția.

FIGURA 2.13

Afișarea unui mesaj la efectuarea unui clic pe **OK** sau pe **Cancel**.



Acum încercați să efectuați clic pe butonul **Press Me** de câteva ori, după care efectuați clic pe **OK** sau pe **Cancel**. De această dată ar trebui să vedeți un mesaj care vă spune de câte ori ați efectuat clic asupra butonului **Press Me** (vedeți figura 2.14). Efectuarea unui clic pe **OK** duce la încheierea programului.

FIGURA 2.14

Testarea mesajului de numărare.

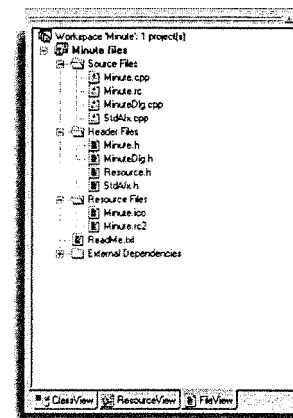


Lucrul în reprezentarea fișierelor

Efectuați un clic pe eticheta de pagină **FileView** aflată la baza ferestrei spațiului de lucru al proiectului. Ca urmare este afișată o listă cu toate fișierele incluse în proiect (vedeți figura 2.15).

FIGURA 2.15

Selectarea paginii **FileView** vă permite să navigați printre fișierele proiectului.



Fișierele sunt organizate pe cataloage în funcție de tipul acestora (cataloagele nu reprezintă locația fișierelor pe hard-disc). Pentru a deschide un fișier în fereastra editorului, efectuați un dublu clic pe numele acestuia. Încercarea de a deschide fișierul Minute.rc va duce la selectarea paginii ResourceView, deoarece fișierul .rc conține informații despre resurse. Pentru o listă a tipurilor uzuale de fișiere din cadrul unui proiect Visual C++, consultați tabelul 2.2.

TABELUL 2.2 Tipuri uzuale de fișiere utilizate în Developer Studio

Extensie	Tip	Descriere
.dsw	Fișier spațiu de lucru	Combină informațiile proiectelor într-un spațiu de lucru.
.dsp	Fișier proiect	Reține informații despre un proiect și înlocuiește fișierul .mak.
.clw	Fișier ClassWizard	Reține informații despre clase utilizate de către instrumentul ClassWizard.
.ncb	Fișier pentru navigare fără compilare	Conține informații utilizate de către ClassView și de către bara cu instrumente ClassWizard.
.opt	Fișier de opțiuni	Reține opțiuni personalizate privind aspectul spațiului de lucru.
.cpp	Fișier sursă C++	Conține cod de implementare.
.h	Fișier antet C++	Conține cod de definiție și de declarație de clase.
.rc	Fișiere script de resurse	Conține informații despre resurse precum casetele de dialog și meniurile.
.rc2	Fișiere script de resurse	Folosit pentru a include resurse în mai multe proiecte.
.bmp	Fișier bitmap	Reține imagini de tip bitmap.
.ico	Fișier pictogramă	Reține imagini de tip pictogramă.

Regenerarea fișierului de informații ClassWizard

Dacă ștergeți fișierul .clw din directorul proiectului, data viitoare când încercați să utilizați ClassWizard veți fi întrebat dacă doriți regenerarea acestui fișier. Acest lucru este necesar uneori, atunci când în caseta de dialog ClassWizard nu sunt accesibile toate clasele sau toți identificatorii de controale.

Această listă nu se vrea câtuși de puțin a fi completă. Dacă accesați directorul unui proiect cu Explorer, veți vedea alte câteva tipuri de fișiere. Unele dintre acestea sunt fișiere temporare folosite de către Developer Studio; altele depind de tipul proiectului și de conținutul său.

Meniul contextual din reprezentarea fișierelor vă permite să efectuați mai multe operații. Puteți să compilați programe individuale sau o selecție de programe din cadrul proiectului.

Acest lucru se dovedește util atunci când lucrați la o anumită parte și nu vă este necesară asamblarea întregului proiect. Puteți, de asemenea, să adăugați la proiect fișiere ale altor proiecte.

Înlăturarea fișierelor din proiect

Pentru a înlătura un fișier din proiect, localizați numele fișierului în arborele FileView și apăsați pur și simplu tasta Delete. Rețineți că aceasta duce doar la înlăturarea fișierului din proiectul Visual Studio, fără a șterge efectiv fișierul de pe disc.

Administrarea proiectelor

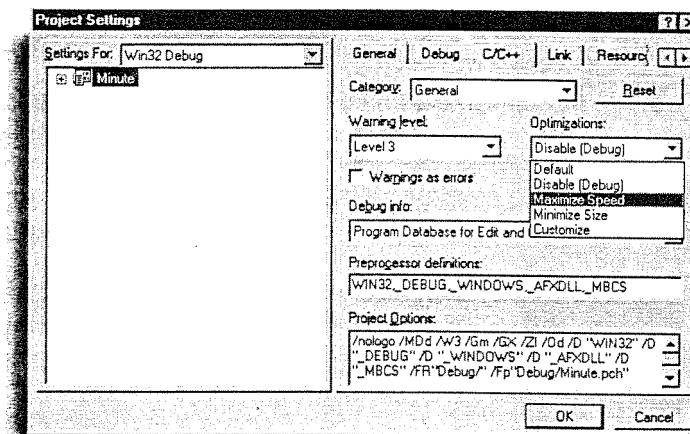
Mai demult, una dintre cele mai fine operații la dezvoltarea de aplicații în C sau C++ era întreținerea fișierului de asamblare. Fișierul de asamblare are scopul de a furniza compilatorului instrucțiuni detaliate privind modul de compilare a programelor și care programe vor fi legate și în ce ordine. Acum Developer Studio se ocupă în culise de această sarcină laborioasă și de cele mai multe ori nici nu veți ști despre asta. De fapt, proiectele Visual C++ 6.0 nu conțin un fișier de asamblare, deși puteți genera un astfel de fișier, dacă doriți, prin selectarea opțiunii **Export Makefile** din meniul **Project**.

Opțiunile de proiect

Puteți să modificați opțiunile de compilare și să efectuați multe alte operații de administrare a proiectului prin intermediul casetei de dialog Project Settings. Pentru a deschide caseta de dialog Project Setting, selectați **Settings** din meniul **Project**. Veți vedea o casetă de dialog paginată, înfățișată aici în figura 2.16.

FIGURA 2.16

Compilarea, legarea și alte facilități sunt disponibile în caseta de dialog Project Settings.



Iată câteva dintre opțiunile utile pe care le aveți la dispoziție:

- În pagina **Debug** puteți să specificați parametri de linie de comandă care să fie utilizați de program la rulare.
- În pagina **C++** puteți să adăugați directive pentru preprocesor și să selectați opțiuni de optimizare ale compilatorului.
- În pagina **Link** puteți să specificați directorul în care va fi plasat fișierul executabil.

Configurații adiționale

Proiectul nostru Minute dispune deja de două configurații – o configurație pentru depanare (Debug) și o configurație pentru versiunea finală (Release). Este posibilă crearea unor configurații proprii pentru un proiect. Spre exemplu, ați putea crea configurații cu diferite opțiuni de optimizare în scopul testării performanțelor. Pentru a crea sau a înlătura o configurație, selectați **Configurations** din meniul **Build**. În marea majoritate a cazurilor, configurațiile implicite Debug și Release se dovedesc suficiente.

CAPITOLUL

3

Crearea și proiectarea casetelor de dialog

Editarea unei casete de dialog existente și crearea unor noi machete de casete de dialog

Cum se stabilesc proprietățile vizuale ale casetelor de dialog și ale controalelor acestora

Utilizarea instrumentelor oferite de editorul de resurse pentru poziționarea și alinierea controalelor

Cum poate fi adăugată o funcționalitate la momentul proiectării casetei de dialog

Caseta de dialog este una dintre cele mai redutabile arme din arsenalul oricărui dezvoltator sub Windows datorită utilității sale într-o multitudine de situații. Casetele de dialog pot fi utilizate pentru simpla afișare de mesaje către utilizator, așa cum ați văzut deja în cazul funcției MFC `MessageBox()`. Dar forma cea mai uzuală în care se folosesc casetele de dialog este cea prin care utilizatorii au posibilitatea să introducă informații pentru aplicație. În mod normal, utilizatorul poate să confirme sau să anuleze datele introduse. Anularea închide pur și simplu caseta de dialog, ca și cum aceasta nici nu ar fi fost deschisă; confirmarea duce la efectuarea unei acțiuni care implică datele introduse, după care, de multe ori, dar nu întotdeauna, închide caseta de dialog.

Crearea casetelor de dialog folosite în aplicațiile Visual C++ se realizează în două etape. Prima etapă este faza de proiectare; aceasta presupune crearea machetei pentru caseta de dialog și adăugarea controalelor necesare. Cea de a doua etapă este faza de programare, care presupune stabilirea de legături între caseta de dialog și controalele sale, pe de o parte, și clasele și funcțiile din codul C++, pe de altă parte. Prima etapă (faza de proiectare) poate fi parcursă fără a scrie nici o linie de cod, și de aceasta ne vom ocupa în capitolul de față.

Crearea de machete pentru casetele de dialog

Deși este posibil să creați și să proiectați casete de dialog în Developer Studio fără a crea în prealabil un proiect, se obișnuiește mai degrabă adăugarea și editarea casetelor de dialog în cadrul unui proiect existent. Mai întâi veți folosi AppWizard pentru a crea un nou proiect corespunzător unei aplicații de tip dialog. Cu această ocazie veți studia câteva dintre opțiunile oferite de AppWizard.

Folosiți AppWizard ca și punct de pornire pentru propriile aplicații

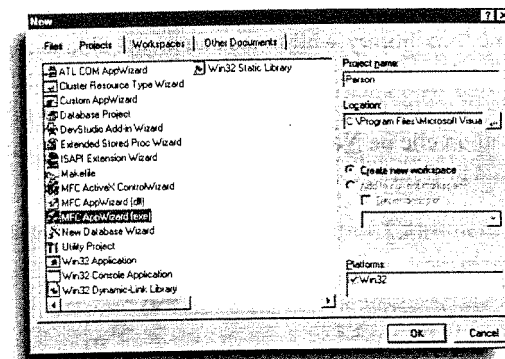
AppWizard este un instrument puternic al cărui scop este generarea unui nou proiect funcțional. AppWizard vă permite să precizați mai multe caracteristici personalizabile pe care le implementează apoi în fișierele sursă pe care le creează. Proiectele generate cu AppWizard sunt complet funcționale; ele pot fi asamblate și rulate fără a fi nevoie de cod adițional.

Crearea unui nou proiect pentru o aplicație de tip dialog

1. Efectuați un clic pe meniul **File** și selectați **New**.
2. Selectați pagina **Projects**; în lista cu tipurile de proiecte selectați **MFC AppWizard (exe)**.
3. Efectuați un clic pe caseta **Project Name** și introduceți numele proiectului: **Person**. Caseta de dialog **New Project** ar trebui să arate ca în figura 3.1.
4. Efectuați un clic pe **OK**. Acum ar trebui să vedeți caseta de dialog **MFC AppWizard – Step 1**.
5. Selectați butonul de opțiune **Dialog Based**.
6. Efectuați un clic pe **Next** pentru a avansa la caseta de dialog **MFC AppWizard – Step 2**.

FIGURA 3.1

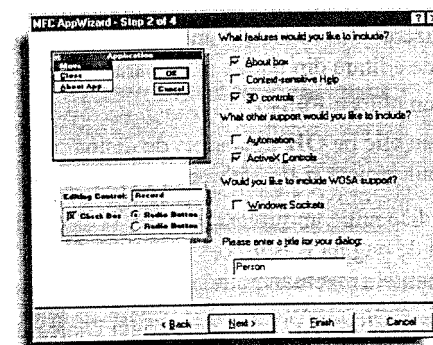
Crearea unui nou proiect.



În acest pas aveți posibilitatea să personalizați anumite elemente ale aplicației chiar înainte ca aceasta să fie creată. Efectul selectării și deselectării diferitelor caracteristici vizuale poate fi văzut în diagramele aflate în partea stângă. Puteți, totodată, să modificați titlul casetei de dialog principale. Alte opțiuni privesc activarea suportului pentru facilități mai avansate. Asigurați-vă că opțiunile sunt stabilite ca în figura 3.2.

FIGURA 3.2

Selectarea opțiunilor privind facilitățile și suportul.



7. Efectuați un clic pe **Next** pentru a trece la caseta de dialog **MFC AppWizard – Step 3**.

Solicitați AppWizard să includă comentarii ajutătoare

Puteți solicita ca AppWizard să includă comentarii în cadrul fișierelor sursă. Acestea vă pot ajuta să înțelegeți codul generat și vă arată unde trebuie plasat codul adițional.

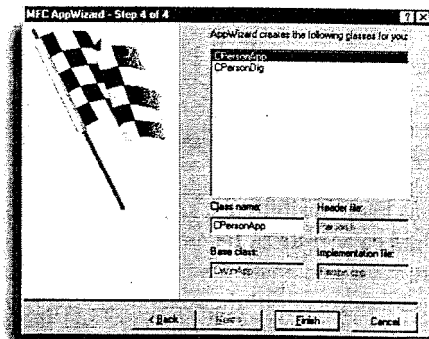
În această etapă sunteți întrebat dacă doriți ca AppWizard să adauge comentarii la codul pe care îl generează. Comentariile sunt de natură explicativă și vă ajută să înțelegeți unde trebuie să adăugați propriul cod. Eu, personal, selectez întotdeauna **Yes, Please** pentru a include comentarii.

În mod implicit, AppWizard creează un program care folosește o versiune DLL (dynamic link library – bibliotecă cu legare dinamică) a MFC. Acest lucru este avantajos deoarece fișierul executabil (.exe) rezultat din proiect va fi mai mic, pentru că toate funcțiile MFC necesare vor fi căutate în alte fișiere de pe disc.

8. Efectuați un clic pe **Next** pentru a trece la caseta de dialog MFC AppWizard – Step 4, înfățișată în figura 3.3.

FIGURA 3.3

O privire asupra claselor și fișierelor generate.



În această ultimă etapă puteți vedea numele claselor și fișierelor care vor fi generate de AppWizard. Dacă aceste nume nu vă satisfac, le puteți modifica prin intermediul casetelor de editare din partea inferioară a casetei de dialog.

9. Efectuați un clic pe **Finish**.
10. Efectuați un clic pe **OK** din caseta de dialog New Project Information, iar AppWizard va crea noul proiect și fișierele sursă ale acestuia.

Acum aveți la dispoziție un proiect și puteți să-i editați resursele. În acest scop, efectuați un clic pe eticheta de pagină **ResourceView** din fereastra spațiului de lucru, după care expandați elementele efectuând clic pe semnele + (plus) alăturat în partea lor stângă.

Veți vedea că AppWizard a creat în catalogul Dialog două machete pentru casete de dialog. Macheta IDD_PERSON_DIALOG reprezintă caseta de dialog principală a aplicației, pe care AppWizard a creat-o din cauză că ați optat pentru o aplicație de tip dialog. Macheta IDD_ABOUTBOX este caseta de dialog apelată prin intermediul opțiunii de meniu **Help About** și afișează pictograma aplicației și numărul de versiune. Pentru început veți efectua o serie de operații de editare simple în caseta de dialog creată de AppWizard; apoi veți vedea cum puteți să adăugați în proiect noi casete de dialog și veți utiliza câteva dintre facilitățile mai avansate oferite de editorul de resurse.

Editarea machetei pentru caseta de dialog principală a aplicației

1. Efectuați un dublu clic pe elementul IDD_PERSON_DIALOG din cadrul secțiunii ResourceView.
2. Efectuați un clic pe textul **TODO: Place Your Controls Here** și apoi apăsați tasta Delete. Textul va fi înlăturat.

3. Efectuați un clic pe butonul **Cancel** și apoi apăsați tasta Delete. Butonul **Cancel** va fi înlăturat.
4. Efectuați un clic pe **OK**.
5. Efectuați un clic cu butonul drept al mouse-ului pentru a apela un meniu contextual.
6. Selectați **Properties** din meniul contextual. Va fi afișată caseta de dialog Push Button Properties.
7. Selectați pagina **General**; apoi înlocuiți cuvântul **OK** din caseta **Caption** cu **Exit**.

Fișierul resource.h

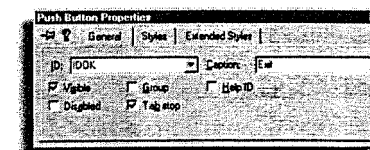
În mod implicit, directivele `#define` pentru ID-urile resurselor din proiect se află în fișierul antet `resource.h`. Acest fișier poate fi vizualizat și editat prin selectarea opțiunii **Resource Symbols** din meniul **View**.

În acest moment ați modificat eticheta unui control fără a afecta modul în care controlul operează în cadrul programului. Programele identifică resursele prin ID. În cazul de față, ID-ul folosit la adăugarea codului care răspunde la mesajele corespunzătoare controlului este `IDOK`. `IDOK` este un ID de control predefinit de către MFC care determină apelarea funcției `CDialog::OnOk()`, aceasta conținând codul necesar pentru închiderea casetei de dialog.

8. Închideți caseta de dialog Push Button Properties. Caseta de dialog ar trebui să arate acum ca în figura 3.4.

FIGURA 3.4

Editarea proprietăților unui control existent.



Ați adus o serie de modificări minore unei casete de dialog existente prin amabilitatea lui AppWizard, dar ce se întâmplă dacă doriți o casetă de dialog complet nouă?

Crearea unei noi machete pentru o casetă de dialog

1. Selectați pagina ResourceView a ferestrei spațiului de lucru al proiectului.
2. Efectuați un clic cu butonul drept pe catalogul Dialog. Va fi afișat un meniu contextual.
3. Selectați **Insert Dialog** din meniul contextual. Va fi creată o nouă machetă de casetă de dialog și aceasta va fi deschisă în fereastra editorului de resurse. Noua casetă de dialog este generată cu butoanele implicite **OK** și **Cancel**.

VEDEȚI...

- Pentru mai multe informații despre pagina ResourceView, vedeți pagina 45.

Stabilirea valorii ID pentru caseta de dialog

La inserarea unei noi machete pentru o casetă de dialog, editorul de resurse asigură atribuirea unui ID unic. Acest ID generat va fi `IDD_DIALOG` urmat de un număr și va figura în catalogul Dialog din secțiunea ResourceView. Primul lucru pe care îl veți face probabil după crearea unei casete de dialog este să înlocuiți ID-ul generat cu ceva mai sugestiv. Spre exemplu, în cazul în care caseta de dialog va fi utilizată pentru a înregistra informații privind vânzările de cărți, un ID posibil ar fi `IDD_CARTI_VANZARI`, care este mult mai ușor de recunoscut decât `IDD_DIALOG9`.

Convenții pentru numele casetelor de dialog

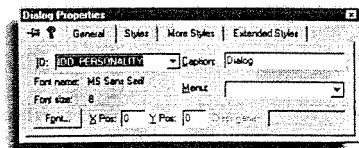
Se obișnuiește prefixarea ID-ului unei casete de dialog cu `IDD_` pentru a face distincția față de celelalte tipuri de resurse.

Modificați ID-ul noii casete de dialog

1. Efectuați un clic cu butonul drept pe suprafața machetei dialogului, în afara oricărui control (efectuați clic pe bara de titlu, în cazul în care caseta de dialog dispune de așa ceva). Va fi afișat un meniù contextual.
Asigurându-vă că nu este selectat nici un control la apelarea meniului de context, veți putea să accesați proprietățile însăși casetei de dialog
2. Selectați **Properties** din meniul contextual. Va fi afișată caseta de dialog Dialog Properties (vedeți figura 3.5).

FIGURA 3.5

Atribuirea unui ID sugestiv pentru o casetă de dialog.



3. Selectați pagina **General** a casetei de dialog Dialog Properties.
4. În cadrul casetei **ID**, înlocuiți `IDD_DIALOG1` tastând `IDD_PERSONALITY`.

Utilizarea proprietăților generale ale casetei de dialog

Modificarea proprietăților este una dintre cele mai uzuale operații din cadrul proiectării casetelor de dialog. Este bine, așadar, să păstrați deschisă caseta de dialog Dialog Properties utilizând pioneza din colțul stânga sus al acesteia, așa cum se vede în figura 3.6. În consecință, caseta de dialog Dialog Properties va rămâne deschisă atunci când efectuați un clic în altă parte, de exemplu pe fereastra editorului de resurse. Developer Studio include și alte câteva casete de dialog cu „pioneze”.

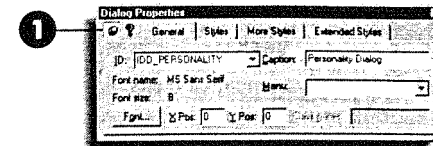
Tipul resursei selectate determină paginile și opțiunile conținute în caseta de dialog Property. Prin intermediul opțiunilor casetei de dialog Property puteți să modificați caracteristicile întregii casete de dialog și ale fiecărui control conținut. Cea de a doua

proprietate pe care este foarte probabil că o veți modifica la crearea unei casete de dialog este titlul acesteia. Titlul casetei de dialog este textul care apare pe bara de titlu.

FIGURA 3.6

Caseta de dialog va rămâne deschisă tot timpul.

- 1 Butonul pioneză este apăsat pentru a menține caseta de dialog deschisă.



Modificați titlu tastând `Personality Dialog` în interiorul casetei **Caption**. Noul titlu ar trebui să apară pe macheta dialogului din editorul de resurse pe măsură ce îl tasteați.

Printre alte opțiuni disponibile în pagina **General** se află modificarea fontului utilizat de către caseta de dialog. Dacă selectați o dimensiune de font mai mare decât 8, veți vedea că întreaga casetă de dialog devine mai mare. Dimensiunea casetei de dialog este calculată pe baza lățimii și înălțimii medii ale fontului utilizat. Fontul implicit este MS Sans Serif de dimensiune 8 și nu se obișnuiește folosirea altor fonturi în casetele de dialog.

Unități de tip dialog

O unitate de tip dialog este numele dat unității de măsură folosită în casetele de dialog și bazată pe fontul asociat dialogului. Este posibil să întâlniți și prescurtarea DLU, care vine de la Dialog Unit – unitate de tip dialog.

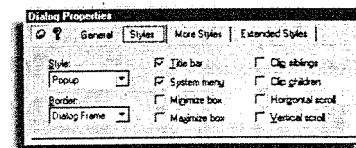
X Pos și **Y Pos** specifică poziția în care va apărea inițial caseta de dialog. Pozițiile sunt relative la fereastra părinte a casetei de dialog. Valori nule pentru aceste coordonate vor determina afișarea casetei de dialog în centrul ferestrei sale părinte.

Utilizarea stilurilor pentru casete de dialog

Veți vedea că în caseta de dialog Dialog Properties se află trei pagini care privesc stilurile. Efectuați un clic pe eticheta **Styles** pentru a selecta prima dintre aceste pagini (vedeți figura 3.7).

FIGURA 3.7

Selectarea stilurilor pentru casete de dialog.



Trebuie să știți că unele opțiuni de stil influențează alte opțiuni. Dacă deselectați stilul Title Bar, aceasta nu va înlătura doar bara de titlu, ci va dezactiva, totodată, opțiunile meniului de sistem și va elimina titlul. Dacă validați apoi **Title Bar**, va trebui să introduceți din nou titlul. O problemă similară poate apare dacă modificați marginea dialogului.

Atenție la opțiunile de stil ale dialogului

Stabilirea unor stiluri de dialog determină automat invalidarea altor opțiuni. În plus, este posibilă stabilirea proprietăților unui dialog astfel încât acesta să devină inutilizabil.

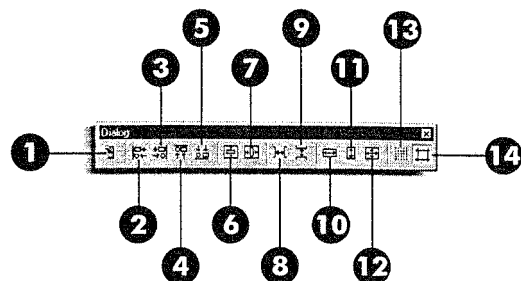
Este bine să știți, de asemenea, că o casetă de dialog poate fi lesne făcută inutilă sau chiar complet invizibilă prin stabilirea unor opțiuni de stil, problemă care poate fi delicat de rezolvat. Dacă suspectați o astfel de situație, cea mai ușoară metodă de a verifica este să comparați opțiunile de stil cu cele ale unei casete de dialog care funcționează corect. În marea majoritate a cazurilor, sunt recomandate opțiunile de stil implicite.

Adăugarea și poziționarea controalelor

Casetele de dialog nu au nici o valoare fără controale. De fiecare dată când deschideți o machetă a unei casete de dialog în cadrul editorului de resurse, în Developer Studio apare un meniu **Layout**. Ar trebui să fie afișate, de asemenea, două noi bare cu instrumente. Una dintre acestea este bara cu instrumente Dialog (vedeți figura 3.8). Cealaltă este bara cu instrumente Controls (vedeți figura 3.9).

FIGURA 3.8

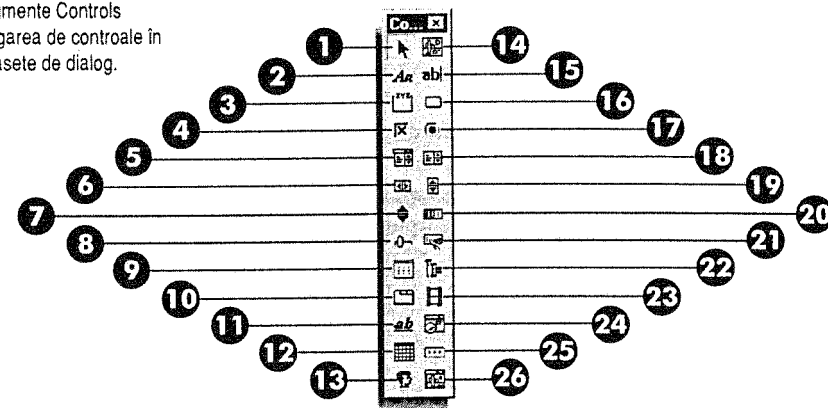
Bara cu instrumente Dialog ajută la poziționarea controalelor.



- | | |
|-----------------------|--------------------------------------|
| 1 Testare | 8 Spațiere orizontală |
| 2 Aliniere la stânga | 9 Spațiere verticală |
| 3 Aliniere la dreapta | 10 Egalarea lățimilor |
| 4 Aliniere în sus | 11 Egalarea înălțimilor |
| 5 Aliniere în jos | 12 Egalarea dimensiunilor |
| 6 Centrare verticală | 13 Comutator pentru grilă |
| 7 Centrare orizontală | 14 Comutator pentru linii de ghidare |

FIGURA 3.9

Bara cu instrumente Controls permite adăugarea de controale în cadrul unei casete de dialog.



- | | |
|--|-------------------------------------|
| 1 Selectare | 14 Control imagine |
| 2 Etichetă statică | 15 Control de editare |
| 3 Etichetă de grup | 16 Control buton |
| 4 Casetă de validare | 17 Buton de opțiune |
| 5 Casetă combinată | 18 Casetă cu listă |
| 6 Bară de derulare orizontală | 19 Bară de derulare verticală |
| 7 Control de modificare incrementală | 20 Control indicator de evoluție |
| 8 Control glisor | 21 Tastă de acces |
| 9 Control listă | 22 Control arbore |
| 10 Control de paginare | 23 Control animație |
| 11 Control casetă de editare formatată | 24 Control de selecție a datei/orei |
| 12 Control calendar | 25 Control adresă IP |
| 13 Control propriu | 26 Casetă combinată extinsă |

Dacă vreuna din barele cu instrumente Dialog sau Controls nu este vizibilă, afișarea ei se face prin una dintre următoarele secvențe de pași:

Afișarea barei cu instrumente Dialog

1. Selectați **Customize** din meniul **Tools**. Este afișată caseta de dialog **Customize**.
2. Selectați pagina **Toolbars**.
3. Selectați **Dialog** din lista **Toolbars**.
4. Efectuați un clic pe **Close**.

Afișarea barei cu instrumente Control

1. Selectați **Customize** din meniul **Tools**. Este afișată caseta de dialog **Customize**.
2. Selectați pagina **Toolbars**.
3. Selectați **Controls** din lista **Toolbars**.
4. Efectuați un clic pe **Close**.

Fiecare imagine de pe bara cu instrumente **Controls** reprezintă un tip distinct de control. Dacă plasați cursorul mouse-ului deasupra unei astfel de imagini, este afișată o casetă mică ce descrie tipul controlului. Selectarea tipului de control care va fi adăugat la caseta de dialog se face prin efectuarea unui clic pe imaginea corespunzătoare. Odată ce ați selectat un control, imaginea sa este afișată în adâncime. Imaginea săgeată din partea stânga sus a barei cu instrumente deselectează controlul și aduce mouse-ul înapoi la comportamentul său obișnuit. Adăugarea unui control într-o casetă de dialog se face selectând mai întâi imaginea corespunzătoare și efectuând apoi clic pe macheta casetei de dialog.

Noua casetă combinată extinsă

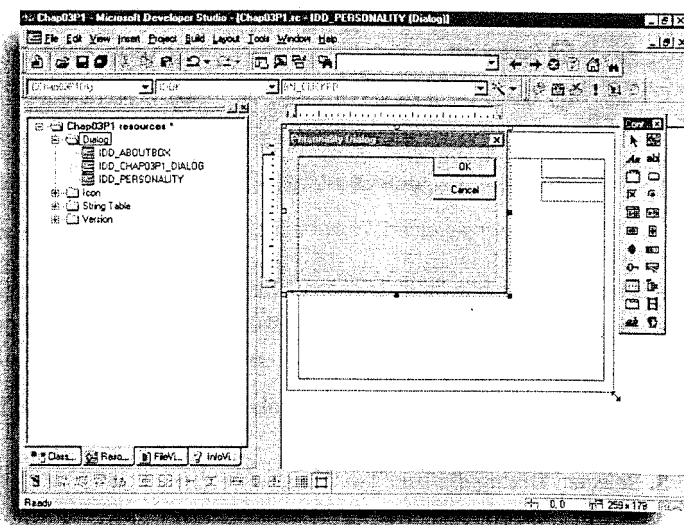
Controlul casetă combinată extinsă vă permite să adăugați imagini la o casetă combinată obișnuită.

Înainte de a adăuga controale în noua casetă de dialog, creșteți dimensiunile acesteia pentru a avea la dispoziție mai mult loc. Modificarea dimensiunilor casetei de dialog se face prin intermediul pașilor de mai jos.

Mărirea casetei de dialog

1. Efectuați un clic pe marginea machetei casetei de dialog. În jurul machetei este afișat un dreptunghi cu puncte de dimensionare, așa cum se vede în figura 3.10.

FIGURA 3.10
Modificarea dimensiunii unei casete de dialog.



2. Efectuați un clic pe punctul de dimensionare din colțul stânga jos și mențineți apăsat butonul mouse-ului.
Puteți modifica oricare dintre dimensiunile casetei de dialog efectuând clic pe punctul de dimensionare adecvat. De exemplu, dacă doriți să modificați dimensiunea casetei de dialog, efectuați un clic pe punctul de dimensionare aflat pe latura din dreapta a casetei de dialog. Cursorul mouse-ului se va modifica pentru a indica dimensiunea sau dimensiunile afectate.
3. Deplasați mouse-ul, menținând butonul său apăsat. Pe măsură ce mouse-ul se mișcă, este afișat conturul casetei de dialog cu noile dimensiuni.
4. Eliberați butonul mouse-ului în momentul în care caseta de dialog are dimensiunile dorite. Casetă de dialog este redesenată.

Acum veți adăuga câteva controale în cadrul casetei de dialog. Pentru a adăuga aceste controale, urmați pașii de mai jos.

Adăugarea de controale în cadrul casetei de dialog

1. Selectați controlul etichetă statică de pe bara cu instrumente **Controls**.
2. Controalele etichetă statică pot fi utilizate pentru simpla afișare de informații textuale. Cel mai adesea, însă, se folosesc pe post de etichete sau titluri pentru alte controale din caseta de dialog.
3. Efectuați un clic înspre partea din stânga sus a casetei de dialog. Pe suprafața acesteia apare un control etichetă statică, conținând textul implicit **Static**.
4. Tastați **First name**. Textul afișat de control se va modifica automat.
5. Selectați controlul de editare de pe bara cu instrumente **Controls**.
6. Efectuați un clic pe suprafața casetei de dialog, în partea dreaptă a etichetei **First name** pe care tocmai ați adăugat-o (poziția exactă nu contează). Este afișat un control de editare.
7. Selectați controlul etichetă statică de pe bara cu instrumente **Controls**.
8. Efectuați un clic pe suprafața casetei de dialog, dedesubtul etichetei **First name**. Este afișat un nou control etichetă statică.

Butonul de comandă implicit

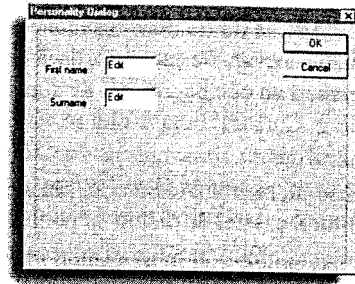
Fiecare casetă de dialog poate conține un buton implicit, a cărui comandă poate fi executată prin apăsarea tastei **Enter**. Butonul implicit este afișat cu un contur îngroșat. Butonul **OK** este de regulă butonul implicit. Un buton poate fi stabilit ca implicit prin intermediul paginii **Styles** a casetei de dialog **Push button Properties**.

9. Tastați **Surname**. Textul afișat de controlul etichetă se modifică automat.
10. Selectați controlul de editare de pe bara cu instrumente **Controls**.
11. Efectuați un clic pe suprafața casetei de dialog, în partea dreaptă a etichetei **Surname** pe care tocmai ați adăugat-o (poziția exactă nu contează). Este afișat un control de editare.

Acum ar trebui să aveți o casetă de dialog având un aspect similar cu caseta din figura 3.11.

FIGURA 3.11

Adăugarea de controale într-o casetă de dialog.



VEDEȚI...

- Pentru mai multe detalii privind controalele buton, vedeți pagina 72.
- Pentru a afla mai multe despre utilizarea controalelor etichetă statică, vedeți pagina 94.
- Pentru amănunte despre controalele de editare, vedeți pagina 100.
- Pentru detalii privind noua casetă combinată extinsă, vedeți pagina 240.

Dimensionarea controalelor

Modificarea dimensiunilor unui control se poate face în mai multe feluri. În primul rând, atunci când tastați un text într-un control etichetă statică, dimensiunea controlului se modifică automat pentru a cuprinde textul introdus. Acest proces este numit *dimensionare după conținut* și este implementat pentru toate controalele care dispun de etichete.

Dimensionarea unui control în funcție de conținut

1. Efectuați un clic asupra controlului. Este afișat un dreptunghi care încadrează controlul.
2. Efectuați un clic cu butonul drept pentru a apela meniul contextual.
3. Selectați **Size to Content**.

Este posibilă modificarea dimensiunii oricărui control prin intermediul punctelor de dimensionare care sunt afișate atunci când un control este selectat. În cazul casetei noastre de dialog, casetele de editare pentru nume ar putea fi ceva mai mari, așa că vom lăți primul control, după care vom vedea cum putem să aducem al doilea control la dimensiunea primului.

Pentru a dimensiona controlul de editare alături de eticheta **First name**, efectuați pașii următori.

Dimensionarea unui control de editare

1. Efectuați un clic pe controlul de editare din partea dreaptă a etichetei **First name**. În jurul controlului va fi afișat un dreptunghi care conține puncte de dimensionare.
2. Deplasați mouse-ul în marginea din dreapta a controlului și poziționați-l deasupra punctului de dimensionare albastru aflat în mijlocul laturii. La atingerea poziției corecte, cursorul mouse-ului va deveni o săgeată stânga-dreapta.

3. Apăsați și țineți apăsat butonul mouse-ului.

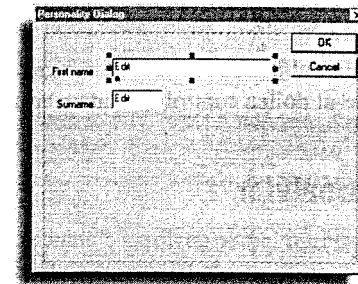
4. Deplasați mouse-ul înspre dreapta, menținând butonul apăsat. Pe măsură ce mișcați mouse-ul este afișat un contur indicând dimensiunea controlului.

5. Eliberați butonul mouse-ului atunci când controlul ajunge la dimensiunea dorită. Controlul este redesenat.

Caseta de dialog va arăta acum similar cu figura 3.12.

FIGURA 3.12

Dimensionarea unui control individual.



Selectarea mai multor controale

Pentru a realiza o interfață îngrijită, este adeseori nevoie ca mai multe controale să fie poziționate unul relativ la altul sau ca mai multe controale să aibă aceeași dimensiune. Așa ceva se poate face selectând mai întâi controalele care trebuie poziționate sau dimensionate și apoi alegând comanda potrivită de dimensionare sau aliniere din cadrul meniului **Layout** sau de pe bara cu instrumente **Dialog**.

Utilizarea unui cadru elastic pentru selectarea mai multor controale

Pentru a selecta mai multe controale de pe suprafața unei machete de dialog puteți folosi ceea ce se numește *cadru elastic*. Cadru elastic se referă la apăsarea butonului mouse-ului și menținerea acestuia apăsat în timp ce cursorul mouse-ului este deplasat peste controalele care se doresc selectate. Pe măsură ce deplasați cursorul este afișat un dreptunghi punctat. La eliberarea butonului mouse-ului vor fi selectate toate controalele aflate în interiorul dreptunghiului.

Este important să rețineți că atunci când doriți să efectuați o operație precum egalizarea dimensiunilor a două controale, controlul asupra căruia ați efectuat ultimul clic va fi controlul selectat (cel cu punctele de dimensionare umplute cu albastru). Controlul selectat este cel care determină ce se întâmplă la rularea unei comenzi de dimensionare sau de aliniere.

Pentru a face ca cel de-al doilea control de editare din caseta noastră de dialog să aibă aceeași lățime cu primul control, urmați pașii de mai jos.

Editarea celui de-al doilea control

1. Efectuați un clic pe controlul de editare din partea dreaptă a etichetei **Surname**. În jurul controlului este afișat un dreptunghi.

Poziționarea mai multor controale

La deplasarea unui control atunci când sunt selectate mai multe controale, vor fi deplasate toate controalele selectate.

- țineți apăsată tasta Ctrl și efectuați un clic pe celălalt control de editare.
În jurul controlului este afișat un dreptunghi. Primul control ar trebui să rămână selectat, dar numai ultimul control selectat dispune de puncte de dimensionare umplute cu albastru.
- Selecționați submeniul **Make Same Size** din meniul **Layout** și apoi selecționați **Same Width**. Cel de-al doilea control de editare ar trebui să aibă acum aceeași lățime cu primul control.

Alinierea controalelor

Pe lângă seria de opțiuni legate de dimensionarea controalelor, meniul **Layout** al editorului de resurse și bara cu instrumente Dialog vă ajută și la alinierea controalelor. Pentru a alinia controale unul relativ la altul trebuie mai întâi să selecționați controalele în cauză (despre aceasta am discutat în secțiunea „Selectarea mai multor controale”, mai devreme în acest capitol). Odată ce ați selectat controalele, pot fi aplicate mai multe operații de aliniere. Vom studia câteva din acestea cu ajutorul casetei noastre de dialog.

Pentru a alinia controalele etichetă statică cu controalele de editare corespunzătoare, efectuați pașii următori.

Alinierea controalelor de editare

- Selecționați controlul First Name și apoi selecționați controlul de editare din partea sa dreaptă. Ambele controale ar trebui să fie încadrate de dreptunghiuri de selecție, dar numai controlul de editare trebuie să aibă puncte de dimensionare umplute cu albastru.
Dacă nu știți exact cum să selecționați mai multe controale, consultați secțiunea „Selectarea mai multor controale” mai devreme în acest capitol.
- Selecționați submeniul **Align** din meniul **Layout** și apoi selecționați **Bottom**. Aceasta va face ca eticheta **First Name** să fie aliniată la marginea inferioară a controlului de editare.
- Repetăți pașii 1 și 2 pentru a doua pereche de controale.
- Acum selecționați controlul First Name și apoi controlul Surname.
- Selecționați submeniul **Align** din meniul **Layout** și apoi selecționați **Right**. Aceasta va face ca marginile din dreapta ale celor două controale etichetă să fie aliniate.

Caseta de dialog ar trebui să arate acum ca în figura 3.13.

Utilizarea liniilor de ghidare

Adăugarea unor linii de ghidare în cadrul machetei casetei de dialog vă poate ajuta la poziționarea controalelor. Liniile punctate albastre pe care le vedeți de-a lungul marginilor casetei de dialog sunt linii de ghidare. Aceste linii de ghidare pot fi observate în figura

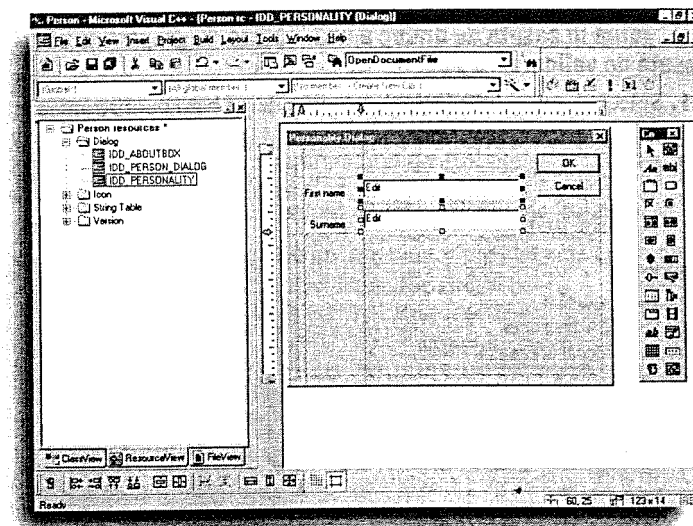
3.13. Atunci când marginea unui control este suprapusă peste o linie de ghidare, deplasarea liniei de ghidare duce la deplasarea controlului. Efectul exercitat de liniile de ghidare poate fi observat prin mărirea lățimii casetei de dialog. Veți vedea că butoanele implicite OK și Cancel își păstrează poziția relativă față de marginea dreaptă a casetei de dialog; aceasta se întâmplă din cauză că sunt legate de margine.

Controalele care ating liniile de ghidare sunt poziționate automat

Atunci când marginea unui control este atașată la o linie de ghidare, controlul în cauză poate fi poziționat prin deplasarea liniei de ghidare. Liniile de ghidare sunt afișate numai la momentul editării resursei, nu și în timpul testării machetei dialogului sau la momentul execuției programului.

FIGURA 3.13

Alinierea marginilor controalelor și utilizarea liniilor de ghidare.



Puteți să adăugați propriile linii de ghidare efectuând clic pe riglele afișate în partea superioară și în stânga casetei de dialog. La adăugarea liniilor de ghidare, în cadrul riglei corespunzătoare apare o săgeată. Poziționarea unei linii de ghidare se face efectuând un clic pe săgeata asociată și deplasând-o de-a lungul riglei. O linie de ghidare adăugată în imediata apropiere a unui control se suprapune automat cu marginea acelui control; în mod similar, un control plasat în imediata apropiere a unei linii de ghidare se aliniază automat cu aceasta.

Organizarea controalelor din casetele de dialog

Pe lângă adăugarea și poziționarea controalelor, există alte câteva aspecte importante care trebuie luate în considerare la crearea casetelor de dialog. Este posibilă folosirea casetelor de grupare pentru un aspect ordonat al casetei de dialog și o mai mare simplitate a acesteia. Se poate specifica ordinea în care controalele sunt accesate la apăsarea tastei Tab; se pot preciza, de asemenea, taste de acces.

Utilizarea casetelor de grupare

O soluție pentru realizarea unei interfețe mai aerisite este gruparea împreună a controalelor, stabilind un titlu pentru grup și trasând un cadru în jurul controalelor. În acest scop se utilizează casetele de grupare. Ca întotdeauna, cel mai bine veți înțelege dintr-un exemplu concret, așa că vom mai adăuga câteva controale în caseta noastră de dialog.

Casetele de grupare nu au ca scop doar o interfață plăcută

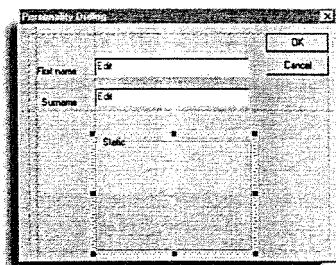
Controalele casetă de grupare nu se folosesc numai pentru a îmbunătăți aspectul unei casete de dialog. Atunci când sunt utilizate împreună cu casete de validare sau cu butoane de opțiune, aceste casete permit asocierea unei mulțimi de controale. Controalele aflate în interiorul fiecărei casete de grupare își pot influența reciproc starea în mod automat.

Adăugarea în caseta de dialog a unei casete de grupare și a mai multor controale casetă de validare

1. Selectați controlul casetă de grupare de pe bara cu instrumente Controls.
2. Adăugați acest control pe suprafața casetei de dialog, sub controalele de editare.
3. Dimensionați caseta de grupare cu ajutorul punctului de dimensionare din colțul stânga jos, așa cum se vede în figura 3.14.

FIGURA 3.14

Gruparea împreună a controalelor în scopul unui aspect mai ordonat al casetei de dialog.



4. Editați eticheta casetei de grupare astfel încât să afișeze Analysis.
5. Selectați controlul casetă de validare de pe bara cu instrumente Controls.
6. Adăugați patru controale casetă de validare. Plasați cele patru controale în interiorul casetei de grupare, așa cum o arată figura 3.15.
7. Editați etichetele casetelor de validare. Puteți să stabiliți orice etichete doriți (eu mi-am dezvoltat propriul model de personalitate).

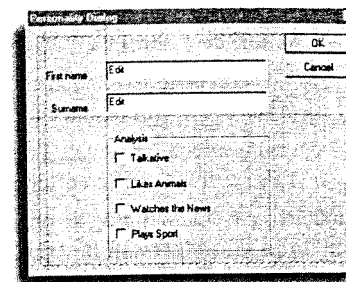
Adăugarea mai multor controale casetă de validare

După plasarea unei casete de validare pe suprafața casetei de dialog trebuie să selectați din nou butonul de pe bara cu instrumente asociat acestui control.

8. Utilizați informațiile din celelalte secțiuni ale acestui capitol care s-au ocupat de dimensionarea, selectarea și alinierea controalelor pentru a aranja controalele din caseta de grupare Analysis într-o manieră ordonată.

FIGURA 3.15

Plasarea controalelor în interiorul unei casete de grupare.



VEDEȚI...

➤ Pentru mai multe informații privind utilizarea casetelor de grupare, vedeți pagina 82.

Stabilirea ordinii de selectare

Controalele dintr-o casetă de dialog pot fi accesate în serie; această serie este cunoscută sub numele de *ordine de selectare*. Pentru a accesa următorul control din serie utilizatorul apasă tasta Tab, iar accesarea controlului precedent se face prin apăsarea combinației Shift+Tab. Atunci când un control este accesat, se spune că primește focusul. Spre exemplu, la accesarea unui control buton veți vedea un dreptunghi punctat care apare pe suprafața butonului. Acesta arată că butonul respectiv deține focusul.

Navigarea între controalele unei casete de dialog

Ordinea de selectare reprezintă ordinea în care controalele unei casete de dialog primesc focusul la apăsarea tastei Tab. Un singur control poate avea focusul la un moment dat.

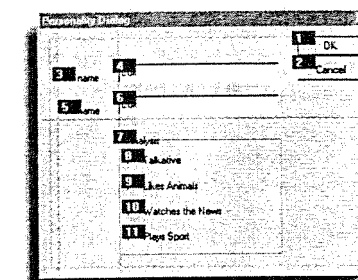
Un control poate fi exclus din ordinea de selectare prin invalidarea proprietății **Tab Stop** din pagina **General** a casetei de dialog Control Property.

Specificarea ordinii de selectare pentru o casetă de dialog

1. Selectați **Tab Order** din meniul **Layout**. Macheta casetei de dialog este afișată împreună cu numărul de ordine al fiecărui control, așa cum se vede în figura 3.16.

FIGURA 3.16

Modificarea ordinii de selectare a controalelor.



2. Efectuați un clic pe controlul care doriți să primească focusul la afișarea inițială a casetei de dialog. Alături de control va fi afișat numărul 1.
 3. Efectuați clic pe toate celelalte controale în ordinea în care doriți să aibă loc selectarea. Fiecare control va primi următorul număr din serie.
 4. Pentru a testa ordinea de selectare, alegeți **Test** din meniul **Layout** sau apăsați Ctrl+T. Apoi folosiți tasta Tab pentru a verifica ordinea de selectare dorită.
- La momentul testării, caseta de dialog se prezintă ca și cum ar fi utilizată în cadrul aplicației, permițându-vă să îi testați funcționalitatea. Pentru a încheia testarea apăsați Esc.

Stabilirea tastelor de acces

O tastă de acces pentru un control se specifică prin inserarea unui simbol & în eticheta controlului înainte de caracterul care va avea rol de mnemonică. De exemplu, dacă un control buton are eticheta `Exit`, eticheta afișată pe buton va fi de fapt `Exit`, iar apăsarea butonului va putea fi simulată prin combinația de taste Alt+X sau Alt+x.

Mnemonicile de acces

O *mnemonică* reprezintă un caracter subliniat care poate fi utilizat ca metodă rapidă de accesare a unui control anume. Apăsarea tastei Alt și a caracterului mnemonic realizează accesul. Mnemonicile sunt specificate în editorul de resurse prin inserarea unui & imediat înainte de litera care se dorește a fi utilizată pentru acces rapid.

Dacă mai multe controale dintr-o casetă de dialog au asociată aceeași mnemonică, la apăsarea combinației de acces va fi selectat dintre acestea primul control în raport cu ordinea de selectare. Pentru a evita confuzii, editorul de resurse vă permite să verificați prezența unor mnemonice duplicate. Pentru a verifica mnemonicele duplicate, selectați **Check Mnemonics** din meniul contextual al editorului de resurse.

În cazul în care nu există mnemonice duplicate va fi afișat mesajul `No duplicate mnemonics found`; în caz contrar, veți fi întrebat dacă doriți selectarea controalelor cu aceeași mnemonică astfel încât să puteți rezolva conflictul.

VEDEȚI...

- Pentru mai multe informații despre adăugarea combinațiilor de taste, vedeți pagina 95.

CAPITOLUL

4

Utilizarea controalelor de tip buton

Utilizarea diferitelor tipuri de butoane

Manipularea butoanelor în timpul execuției

Maparea mesajelor peste funcții C++

Există trei tipuri de butoane: butoane de comandă, butoane de opțiune și casete de validare. Fiecare tip are propria funcționalitate. Un buton de comandă reprezintă o comandă care se execută la efectuarea unui clic asupra butonului. Butoanele de opțiune permit selectarea unui element dintr-o listă de opțiuni care se exclud reciproc. La un moment dat nu poate fi selectat decât un singur element dintr-un grup de butoane de opțiune. Casetele de validare sunt folosite pentru a permite o opțiune în sensul selectării unui element; spre deosebire de butoanele de opțiune, la un moment dat pot fi selectate mai multe casete de validare (sau nici unul) dintr-un grup.

Butoanele sunt controalele utilizate cel mai frecvent. Le veți întâlni practic în toate casetele de dialog. Folosirea corectă a fiecărui tip de buton înlesnește substanțial înțelegerea și utilizarea unui program.

Utilizarea butoanelor de comandă

Butonul de comandă este cel mai simplu control din Windows. Fiecare buton de comandă reprezintă o singură comandă și, la efectuarea unui clic asupra sa, îndeplinește acțiunea care execută acea comandă. Aproape toate casetele de dialog conțin cel puțin două butoane de comandă: OK și Cancel.

CDialog implementează funcții de tratare implicite pentru OK și Cancel

Editorul de resurse adaugă în casetele de dialog nou create butoanele de comandă OK și Cancel. Clasa MFC de bază pentru toate casetele de dialog este CDialog, iar aceasta conține implementări implicite care răspund la mesajele BN_CLICKED ale acestor butoane: OnOK() și OnCancel(). Puteți să supradefiniți aceste funcții în propriile clase dialog derivate din CDialog pentru a le ajusta comportamentul.

AppWizard creează automat aceste butoane pentru aplicațiile de tip dialog. Multe casete de dialog implică și alte butoane de comandă; ați văzut că acestea pot fi adăugate prin intermediul editorului de resurse. Proprietățile și opțiunile de stil inițiale pentru fiecare buton sunt specificate tot prin intermediul editorului de resurse, dar ce se întâmplă dacă doriți modificarea acestor proprietăți la momentul execuției? De pildă, ați putea dori să modificați eticheta unui buton în funcție de informații disponibile numai după lansarea aplicației. În acest scop aveți nevoie de acces la control în cadrul programului și de o metodă cu ajutorul căreia să specificați noua etichetă. În exemplul următor veți vedea cum funcția GetDlgItem() este folosită pentru a obține un pointer la un obiect CWnd care reprezintă controlul buton. Apoi veți vedea cum se folosește pointerul CWnd pentru a modifica eticheta și alte atribute în timpul rulării programului.

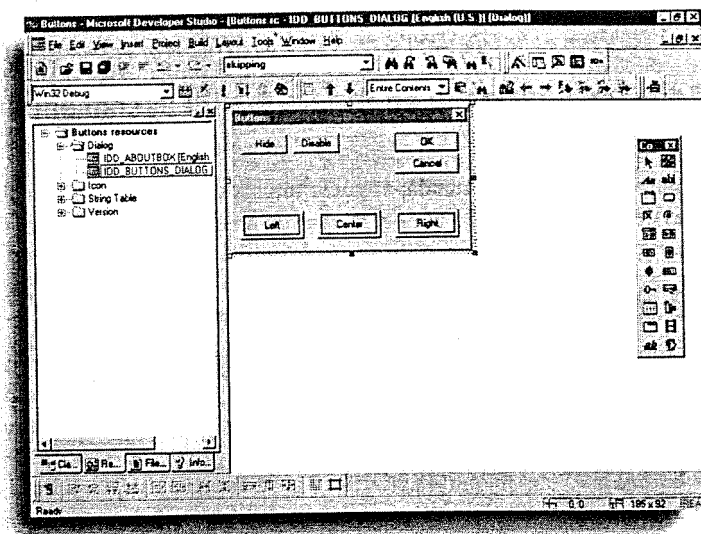
Utilizați AppWizard pentru a crea o nouă aplicație de tip dialog numită Buttons. Odată ce ați creat noul proiect, va trebui să adăugați cinci butoane în caseta de dialog Buttons. Pe parcursul operației următoare s-ar putea dovedi necesar să sporiți lățimea casetei de dialog și să modificați dimensiunile butoanelor.

Adăugarea butoanelor în caseta de dialog Buttons

1. Selectați pagina **ResourceView**, expandați catalogul **Dialog** și efectuați un dublu clic pe elementul **IDD_BUTTONS_DIALOG**. Caseta de dialog este afișată în cadrul editorului de resurse. Efectuați un clic pe eticheta **TODO: Place dialog controls here** și apoi apăsați tasta Delete pentru a o înlătura.
2. Selectați pictograma Button de pe bara cu instrumente Controls și adăugați două butoane în partea superioară stânga a casetei de dialog, așa cum se vede în figura 4.1.

FIGURA 4.1

Aspectul casetei de dialog Buttons.



3. Selectați primul dintre aceste butoane, efectuați un clic cu butonul drept și selectați **Properties** din meniul contextual. Este afișată caseta de dialog Push Button Properties. Puteți menține caseta de dialog cu proprietăți vizibilă prin efectuarea unui clic pe imaginea de pionieră din colțul stânga sus al acesteia.
4. Introduceți un nume în caseta combinată **ID**; pentru exemplul de față folosiți numele **IDC_SHOW_HIDE**. Introduceți o etichetă în caseta **Caption**; pentru exemplul de față folosiți eticheta **Hide**.
5. Selectați cel de al doilea buton. Introduceți **IDC_ENABLE_DISABLE** în caseta combinată **ID** și **Disable** în caseta **Caption**.
6. Adăugați alte trei butoane înspre partea inferioară a casetei de dialog (vedeți figura 4.1).
7. Selectați cele trei butoane pe care tocmai le-ați adăugat prin efectuarea de clic pe fiecare în timp ce țineți apăsată tasta Ctrl.
8. Efectuați un clic pe pagina **Extended Styles** a casetei de dialog Multiple Selection Properties. Selectați opțiunile **Client Edge** și **Modal Frame**.

9. Selectați butonul cel mai din stânga. Introduceți `IDC_LEFT` pentru **ID** și `Left` pentru **Caption**. Selectați butonul din mijloc. Introduceți `IDC_CENTER` pentru **ID** și `Center` pentru **Caption**. Selectați butonul din dreapta. Introduceți `IDC_RIGHT` pentru **ID** și `Right` pentru **Caption**.
10. Închideți caseta de dialog Push Button Properties. Caseta de dialog ar trebui să arate acum ca în figura 4.1.

VEDEȚI...

- Pentru detalii despre modul de creare a unei aplicații de tip dialog, consultați pagina 55.
- Pentru mai multe informații despre proiectarea casetelor de dialog, vedeți pagina 60.

Adăugarea rutinelor de tratare pentru evenimentele clic ale butoanelor

Controlul buton de comandă, asemeni tuturor celorlalte controale din Windows, trimite mesaje către fereastra sa părinte la apariția unui eveniment. Atunci când se efectuează clic asupra unui buton de comandă dintr-o casetă de dialog, către aceasta este transmis mesajul `BN_CLICKED`. Pentru ca programul să răspundă printr-o acțiune adecvată, este necesară crearea unei funcții de tratare pentru mesajul `BN_CLICKED`.

La finalul exemplului Buttons va exista câte o funcție de tratare pentru fiecare din cele cinci butoane. Funcția asociată butonului `IDC_SHOW_HIDE` va comuta vizibilitatea celor trei butoane de la baza casetei de dialog și va comuta eticheta acestui buton între **Show** și **Hide**. Funcția asociată butonului `IDC_ENABLE_DISABLE` va comuta starea de activare a celor trei butoane din partea inferioară și va comuta eticheta acestui buton între **Enable** și **Disable**. Atunci când un buton este dezactivat, eticheta sa este estompată, sau voalată, și butonul nu poate recepționa acțiuni de la mouse sau de la tastatură.

Butoanele **Left**, **Right** și **Center** vor răspunde toate prin modificarea titlului casetei de dialog.

În continuare veți adăuga funcția de tratare pentru `BN_CLICKED` asociată primului buton, după care veți trece la secțiunea următoare, care se ocupă de scrierea codului și de adăugarea funcțiilor de tratare pentru celelalte butoane.

Adăugarea unei funcții de tratare a clicului asupra unui buton

1. Efectuați un dublu clic pe butonul **Hide**. Va fi afișată caseta de dialog Add Member Variable.
2. Efectuați un clic pe **OK** pentru a accepta numele implicit al funcției membru, `OnShowHide`. Scheletul noii funcții va fi creat și va fi afișat în fereastra editorului, gata pentru adăugarea de cod.

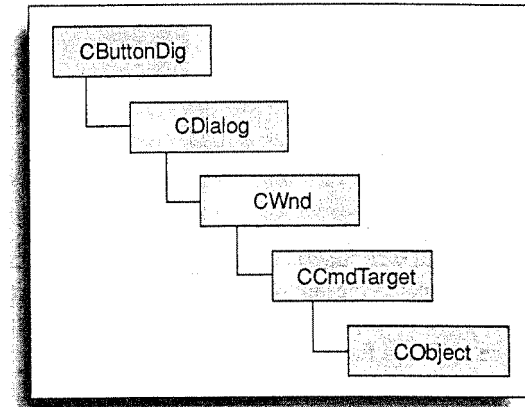
`OnShowHide()` va fi apelată de fiecare dată când se efectuează un clic pe butonul `IDC_SHOW_HIDE`, dar cum anume se întâmplă acest lucru? Pentru a răspunde la întrebare va trebui să divaghez puțin și să explic câteva lucruri legate de hărțile de mesaje.

Despre hărțile de mesaje

Infrastructura MFC utilizează un sistem numit mapare de mesaje pentru a face legătura între mesajele Windows și funcții C++ specifice. O *mapare de mesaje* poate fi adăugată în orice clasă derivată din clasa MFC `CCmdTarget`. Ierarhia care conduce la clasa `CButtonsDlg` este ilustrată în figura 4.2.

FIGURA 4.2

Ierarhia care conduce la clasa `CButtonsDlg`.



Mapările de mesaje sunt realizate de CCmdTarget

O mapare de mesaje este un mecanism care determină ce funcție C++ va fi apelată ca răspuns la un mesaj din Windows. Pentru a putea implementa o mapare de mesaje, o clasă trebuie să fie derivată din `CCmdTarget`.

O bună parte din codul aflat în spatele mapărilor de mesaje este ascuns în macro-uri, care sunt adăugate și actualizate de către instrumentele Visual C++. Definiția clasei `CButtonsDlg` (din fișierul `ButtonsDlg.h`) conține următoarea linie:

```
DECLARE_MESSAGE_MAP()
```

AppWizard a inserat această linie la crearea clasei. Macro-ul declară de fapt o serie de variabile adiționale pentru a întreține intrări în cadrul *mapărilor de mesaje*, precum și funcții care conduc mesajele către funcția C++ corespunzătoare.

La crearea funcției de tratare `OnShowHide()` s-au întâmplat următoarele trei lucruri:

1. A fost creată implementarea funcției, care arată astfel:

```

void CButtonsDlg::OnShowHide()
{
    // TODO: Add your control notification handler code here
}
  
```

2. A fost adăugată declarația funcției (în `CButtonsDlg.h`), care arată astfel:

```
afx_msg void OnShowHide();
```

Prefixul `afx_msg` nu este nimic deosebit; rolul lui este să amintească faptul că aceasta este o funcție de tratare a unui mesaj.

3. A fost adăugată o intrare în macro-ul `BEGIN_MESSAGE` pentru a stabili legătura dintre mesaj și funcție. Acest macro, aflat în fișierul `CButtonsDlg.cpp`, arată astfel:

```
1 BEGIN_MESSAGE_MAP(CButtonsDlg, CDialog)
2 //{AFX_MSG_MAP(CButtonsDlg)
3 ON_WM_SYSCOMMAND()
4 ON_WM_PAINT()
5 ON_WM_QUERYICON()
6 ON_BN_CLICKED(IDC_SHOW_HIDE, OnShowHide)
7 //}AFX_MSG_MAP
8 END_MESSAGE_MAP()
```

Puteți vedea în linia 6 că funcția `OnShowHide()` are alocată o intrare în cadrul macro-ului care realizează maparea mesajelor. Atunci când Windows trimite casetei de dialog un mesaj `BN_CLICKED` corespunzător butonului `IDC_SHOW_HIDE`, sunt parcurse intrările existente și se apelează funcția de tratare a mesajului, în cazul în care aceasta este găsită. AppWizard a creat celelalte intrări, care corespund unor tipuri diferite de mesaje; acestea și alte tipuri de mesaje vor fi discutate în capitole următoare.

Comentarii speciale inserate de ClassWizard

Comentariul special `//{{AFX_MSG_MAP` este utilizat de instrumente din C++, precum ClassWizard, pentru a înțelege structura codului. Dacă înlăturăm aceste comentarii sau dacă modificăm inadecvat codul aflat între ele, este posibil ca ClassWizard să nu mai funcționeze.

VEDEȚI...

➤ Pentru a afla mai multe despre răspunsul la mesaje din Windows, vedeți pagina 169.

Modificarea butoanelor de comandă la momentul execuției

Editorul de resurse este foarte flexibil în ceea ce privește proiectarea aspectului unei casete de dialog, dar uneori această flexibilitate nu este suficientă. Spre exemplu, dacă ar trebui să dezvoltăm o aplicație care să permită proiectarea schemelor electronice, una dintre necesități ar fi posibilitatea de personalizare a butoanelor. Pentru a modifica proprietățile sau stilul unui buton de comandă (sau ale unui alt tip de control) în timp ce programul rulează trebuie să aveți acces la obiectul care stă în spatele controlului. Acest lucru este ușor de realizat în cazul casetelor de dialog, apelându-se funcția `GetDlgItem()` și transmițând ID-ul controlului în cauză. Funcția va întoarce un pointer către obiectul `CWnd` care reprezintă acel control; pointerul poate fi convertit la o clasă corespunzătoare și apoi utilizat pentru a inspecta și modifica atributele controlului.

`GetDlgItem()` întoarce un pointer `CWnd` din cauză că un buton reprezintă de fapt o fereastră. Aspectul unui buton diferă enorm de cel al unei casete de dialog din cauza unui stil propriu, dar ambele componente sunt ferestre și ambele pot fi reprezentate de un obiect `CWnd`.

Clasa `CWnd` implementează funcționalitatea de bază a ferestrelor

Clasa `CWnd` încapsulează o fereastră și implementează funcționalitatea de bază a acesteia. Este important să rețineți că obiectul `CWnd` este un obiect C++, și nu fereastra propriu-zisă. Fereastra din Windows este o entitate distinctă care se accesează prin intermediul funcțiilor membre ale `CWnd`. Obiectul `CWnd` este construit înainte de crearea ferestrei și fereastra poate fi distrusă apelând `DestroyWindow()` în timp ce obiectul `CWnd` rămâne valid.

Pentru ca funcția de tratare `OnShowHide()` să comute vizibilitatea celor trei butoane din partea inferioară a casetei de dialog, introduceți codul prezentat în listingul 4.1 după comentariile `//TODO`.

LISTINGUL 4.1 LST04_1.CPP – Comutarea vizibilității butoanelor cu `ShowWindow()`

```
1 void CButtonDlg::OnShowHide()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Verificam daca butonul Left este vizibil
6     BOOL bVisible = GetDlgItem(IDC_LEFT)->1
        IsWindowVisible();
7
8     // ** Comutam vizibilitatea fiecarui control
9     GetDlgItem(IDC_LEFT)->ShowWindow(bVisible-2
        ? SW_HIDE : SW_SHOW);
10    GetDlgItem(IDC_CENTER)->ShowWindow(bVisible
        ? SW_HIDE : SW_SHOW);
11    GetDlgItem(IDC_RIGHT)->ShowWindow(bVisible
        ? SW_HIDE : SW_SHOW);
12
13    // ** Comutam eticheta show/hide a butonului
14    GetDlgItem(IDC_SHOW_HIDE)->SetWindowText(bVisible-3
        ? "Show" : "Hide");
15 }
```

- ① `GetDlgItem()` întoarce un pointer la obiectul `CWnd`. `IsWindowVisible()` este o funcție membru a `CWnd` care întoarce `TRUE` dacă fereastra în cauză este vizibilă.
- ② Vizibilitatea fiecărui buton este comutată apelând `ShowWindow()` și transmițând unul din indicatorii `SW_SHOW` sau `SW_HIDE`.
- ③ `SetWindowText()` este o altă funcție membru foarte utilă a `CWnd` care, în acest caz, modifică eticheta butonului.

În apelul funcției `GetDlgItem()` din linia 6 este transmis identificatorul de resursă `IDC_LEFT`, așa că funcția întoarce un pointer către obiectul `CWnd` corespunzător butonului

Left. Acest pointer este apoi folosit pentru a apela funcția `IsWindowVisible()`, care va întoarce `TRUE` sau `FALSE`, după cum este cazul. Rezultatul este depus în variabila locală `bVisible` de tip `BOOL`. Deoarece cele trei butoane sunt vizibile toate sau nici unul, este suficient să verificăm unul singur dintre ele.

În liniile 9, 10 și 11 este apelată funcția `GetDlgItem()` pentru a obține pe rând un pointer `CWnd` la cele trei butoane. Pentru fiecare control se apelează funcția `ShowWindow()` și (în funcție de valoarea `bVisible`) se transmite fie `SW_HIDE`, care ascunde fereastra, fie `SW_SHOW`, care face fereastra vizibilă. Efectul obținut este cel dorit, adică cele trei butoane dispar împreună printr-un singur clic și reapar la un clic următor.

Utilizarea pointerului întors de către `GetDlgItem()`

S-ar putea ca pointerul `CWnd` întors de `GetDlgItem()` să nu fie valid prea multă vreme. Este de preferat să nu rețineți acest pointer; în schimb, apelați din nou funcția pentru a obține adresa de memorie corespunzătoare. Funcția `GetDlgItem()` întoarce `NULL` în cazul în care nu există nici o fereastră cu ID-ul transmis.

Linia 14 apelează din nou funcția `GetDlgItem()`, transmițând de această dată `IDC_SHOW_HIDE`, care este ID-ul butonului pe care s-a efectuat clic. `SetWindowText()` este funcția membru a `CWnd` care permite modificarea etichetei butonului. Șirul transmis este `Show` sau `Hide`, în funcție de valoarea `bVisible`.

La prima vedere ar părea că eticheta va fi **Show** atunci când butoanele sunt vizibile și **Hide** când acestea sunt ascunse, ceea ce ar fi exact pe dos. Observați că valoarea `bVisible` a fost stabilită la începutul funcției, în linia 6; dacă butoanele erau vizibile la acel moment, în linia 14 ele sunt invizibile și eticheta trebuie să fie **Show**.

Introduceți codul prezentat, asamblați și rulați programul și apoi efectuați un clic pe butonul **Hide**. Aceasta ar trebui să provoace dispariția butoanelor **Left**, **Right** și **Center** și înlocuirea etichetei **Hide** a butonului pe care ați efectuat clic cu **Show**. Butoanele ar trebui să reapară la efectuarea unui nou clic.

Acum este nevoie de o funcție de tratare similară care să comute cele trei butoane între starea de activare și cea de dezactivare.

Adăugați o rutină de tratare pentru mesajul `BN_CLICKED` corespunzător butonului `IDC_ENABLE_DISABLE`. Pentru informații ajutătoare consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton” prezentată mai devreme în acest capitol (nu uitați să selectați butonul **Disable**).

Acum ar trebui să aveți o funcție membru numită `OnEnableDisable()`. Pentru ca această funcție să comute starea de activare a celor trei butoane va trebui să introducă codul din listingul 4.2. Puteți observa imediat similitudinile dintre această funcție și funcția `OnShowHide()` din listingul 4.1, așa că ne vom opri asupra diferențelor existente.

LISTINGUL 4.2 LST04_2.CPP – Comutarea stării de activare a butoanelor cu `EnableWindow()`

```
1 void CButtonDlg::OnEnableDisable()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Obținem starea curentă a butonului Left
6     BOOL bState = GetDlgItem(IDC_LEFT)->IsWindowEnabled(); ❶
7
8     // ** Comutam starea fiecărui control
9     GetDlgItem(IDC_LEFT)->EnableWindow(!bState); ❷
10    GetDlgItem(IDC_CENTER)->EnableWindow(!bState);
11    GetDlgItem(IDC_RIGHT)->EnableWindow(!bState);
12
13    // ** Comutam eticheta Enable/Disable a butonului
14    GetDlgItem(IDC_ENABLE_DISABLE)->SetWindowText(bState ❸
        ? "Enable" : "Disable");
15 }
```

❶ `GetDlgItem()` întoarce un pointer la obiectul `CWnd`. `IsWindowEnabled()` este o funcție membru a `CWnd` care întoarce `TRUE` dacă fereastra în cauză este capabilă să răspundă la clicuri.

❷ Starea de activare a fiecărui buton este comutată apelând `EnableWindow()` și transmițând unul din indicatorii `TRUE` sau `FALSE`.

❸ Eticheta butonului `IDC_ENABLE_DISABLE` este stabilită corespunzător prin intermediul `SetWindowText()`.

În linia 6 este apelată funcția `IsWindowEnabled()`, care întoarce starea de activare curentă a butonului `IDC_LEFT`.

În liniile 9, 10 și 11 este apelată funcția `EnableWindow()`, fiindu-i transmis opusul stării curente. Efectul obținut este cel dorit, adică activarea sau dezactivarea fiecărui control.

În linia 14 se obține un pointer la butonul `IDC_ENABLE_DISABLE` prin intermediul funcției `GetDlgItem()`, pointer folosit în apelul `SetWindowText()` cu scopul de a modifica eticheta butonului.

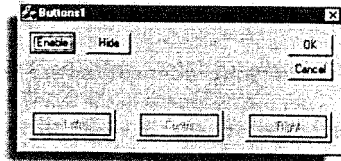
Acum asamblați și rulați programul. Încercați să efectuați un clic pe butonul **Disable**. Aceasta ar trebui să provoace dezactivarea butoanelor **Left**, **Right** și **Center**, așa cum o arată figura 4.3. Un buton dezactivat nu poate primi comenzi de la mouse sau de la tastatură. Veți vedea că starea de activare a unui buton poate fi stabilită chiar și atunci când butonul nu este vizibil.

Pentru a completa exemplul Buttons este nevoie de funcții de tratare pentru cele trei butoane rămase. Fiecare astfel de funcție va conține o singură linie de cod, un apel la o altă funcție numită `ChangeDialogTitle()`. Aceasta modifică titlul casetei de dialog pentru a arăta pe care buton s-a efectuat clic. În plus, ea reține ID-ul butonului pentru ca

titlul dialogului să arate dacă s-au efectuat două clicuri asupra butonului. Listingul 4.3 prezintă întreg codul necesar.

FIGURA 4.3

Butoane de dezactivare



Adăugați rutine de tratare pentru mesajul BN_CLICKED corespunzătoare butoanelor IDC_LEFT, IDC_CENTER și IDC_RIGHT. Pentru informații ajutătoare consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton” prezentată mai devreme în acest capitol.

Adăugarea de variabile și funcții membru prin intermediul ClassView

1. Selectați pagina ClassView a ferestrei spațiului de lucru al proiectului.
2. Efectuați un clic cu butonul drept pe clasa CButtonsDlg pentru a apela meniul contextual al clasei.
3. Selectați Add Member Variable pentru a deschide caseta de dialog Add Member Variable.
4. Introduceți UINT în câmpul Variable Type; apoi apăsați tasta Tab pentru a accesa câmpul următor.
5. Introduceți m_nIDofLastButton în câmpul Variable Declaration; apoi efectuați un clic pe OK.
6. Efectuați un clic cu butonul drept pe clasa CButtonsDlg pentru a apela meniul contextual al clasei.
7. Selectați Add Member Function pentru a deschide caseta de dialog Add Member Function.
8. Introduceți void în câmpul Function Type; apoi apăsați tasta Tab pentru a accesa câmpul următor.
9. Introduceți ChangeDialogTitle(UINT nID) în câmpul Function Declaration; apoi efectuați un clic pe OK.

Acum aveți în clasa asociată casetei de dialog o variabilă care va reține ID-ul butonului pe care s-a efectuat clic. Aveți, de asemenea, o funcție care primește un parametru UINT (întreg fără semn). Introduceți codul celor patru funcții așa cum este prezentat în listingul 4.3.

LISTINGUL 4.3 LST04_3.CPP – Modificarea titlului dialogului pentru a reflecta mesajele de clic asupra butoanelor

```
1 void CButtonsDlg::OnLeft() — 1
2 {
3     // TODO: Add your control notification handler code
```

```
4     ChangeDialogTitle(IDC_LEFT);
5 }
6
7 void CButtonsDlg::OnCenter() — 2
8 {
9     // TODO: Add your control notification handler code
10    ChangeDialogTitle(IDC_CENTER);
11 }
12
13 void CButtonsDlg::OnRight() — 3
14 {
15     // TODO: Add your control notification handler code
16    ChangeDialogTitle(IDC_RIGHT);
17 }
18
19 void CButtonsDlgChangeDialogTitle(UINT nID) — 4
20 {
21     CString strCaption;
22
23     // ** Citim eticheta butonului pe care s-a efectuat clic
24    GetDlgItem(nID)->GetWindowText(strCaption);
25
26    strCaption += " was pressed"; — 5
27
28    // ** Verificam daca este acelasi butonul cu cel de data trecuta
29    if(m_nIDofLastButton == nID)
30        strCaption += " again";
31
32    // ** Stabilim titlu casetei de dialog
33    SetWindowText(strCaption); — 6
34
35    // ** Retinem ID-ul butonului pe care s-a efectuat clic
36    m_nIDofLastButton = nID; — 7
37 }
```

- 1 OnLeft() este funcția de tratare asociată butonului Left care este apelată ca răspuns la mesajul BN_CLICKED.
- 2 OnCenter() este funcția de tratare asociată butonului Center care este apelată ca răspuns la mesajul BN_CLICKED.
- 3 OnRight() este funcția de tratare asociată butonului Right care este apelată ca răspuns la mesajul BN_CLICKED.
- 4 ChangeDialogTitle() nu este o funcție de tratare, ci este apelată de către cele trei funcții de tratare. Ea primește ID-ul butonului pe care s-a efectuat clic.
- 5 Operatorul += din CString atașează text la conținutul existent al variabilei strCaption.

⑥ `SetWindowText()` modifică textul afișat în bara de titlu a casetei de dialog.

⑦ ID-ul butonului pe care s-a efectuat clic este reținut într-o variabilă membru pentru a fi utilizat mai târziu.

Puteți observa în liniile 4, 10 și 16 că fiecare funcție de tratare apelează funcția `ChangeDialogTitle()`. Fiecare dintre aceste apeluri transmite funcției ID-ul butonului în cauză.

În linia 24 este transmis ID-ul funcției `GetDlgItem()`, care întoarce un pointer la obiectul `CWnd` al controlului în cauză. Acest pointer este folosit apoi pentru a apela `GetWindowText()`, ceea ce extrage eticheta butonului. Eticheta este depusă în variabila `strCaption` de tip `CString`. În linia 26, operatorul `+=` al clasei `CString` este folosit pentru a atașa șirul "was pressed" la sfârșitul etichetei.

În linia 29, ID-ul butonului pe care s-a efectuat clic este comparat cu variabila membru `m_nIDofLastButton` (care reține ID-ul ultimului buton pe care s-a efectuat clic); dacă cele două ID-uri sunt egale, în linia 30 se atașează cuvântul "again" la conținutul variabilei `strCaption`.

În linia 33, variabila `strCaption` este transmisă în apelul funcției `SetWindowText()`. Deoarece obiectul `CButtonsDlg` este derivat din `CWnd`, efectul este apelarea funcției `SetWindowText()` pentru caseta de dialog, ceea ce modifică textul afișat în bara de titlu.

În linie 36, ID-ul butonului pe care s-a efectuat clic este depus în variabila membru `m_nIDofLastButton`.

Acum asamblați și rulați programul. Testați-l efectuând clic pe butonul **Center**. Titlul casetei de dialog ar trebui să devină **Center was pressed**. Efectuați încă un clic pe butonul **Center** și titlul dialogului va deveni **Center was clicked again**.

Utilizarea butoanelor de opțiune

Există numeroase situații în care logica presupune alegerea unei singure opțiuni dintre două sau mai multe. De pildă, este de bun simț ca selectarea opțiunii Femeie dintr-un program să ducă la deselectarea automată a opțiunii Bărbat. Butoanele de opțiune sunt create tocmai în această idee și sunt capabile să efectueze automat deselectarea. Butoanele de opțiune sunt plasate aproape întotdeauna în casete de grupare, ceea ce face posibil ca o casetă de dialog să aibă mai multe grupuri de butoane de opțiune care să funcționeze independent.

Deoarece selectarea și deselectarea butoanelor dintr-un grup se petrece în mod automat, singurul lucru pe care trebuie să-l facă programul (atunci când este necesar) este să afle care dintre opțiunile ce se exclud mutual este selectată. În acest scop butoanele de opțiune sunt asociate cu o variabilă. Toate acestea se pot face relativ simplu – iar pentru a ne convinge vom folosi un alt exemplu de program. Apelați la AppWizard pentru a crea un nou proiect de tip dialog cu numele **CityBreak**.

VEDEȚI...

➤ Pentru detalii privind crearea unei aplicații de tip dialog, vedeți pagina 54.

Adăugarea grupurilor de butoane de opțiune

Lucrul cel mai important care trebuie urmărit la crearea de butoane de opțiune este asigurarea corectitudinii proprietăților și a opțiunilor de stil pe care le specificați în cadrul editorului de resurse. Dacă acestea sunt corespunzătoare, cea mai mare parte a funcționalității butoanelor de opțiune vine de la sine.

Utilizarea butoanelor de opțiune cu stilul Auto

La efectuarea unui clic pe un buton de opțiune care are stilul Auto, butonul respectiv este selectat și toate celelalte butoane de opțiune din grupul său și care au același stil sunt în mod automat deselectate.

Pentru fiecare control buton de opțiune dintr-o casetă de grupare trebuie validată opțiunea **Auto Style** pentru a beneficia de deselectare automată. Această opțiune este stabilită implicit la crearea butoanelor de opțiune, așa că nu trebuie să vă preocupați. Gruparea împreună a controalelor se face prin intermediul proprietății **Group**. Un control care are proprietatea **Group** validată este considerat primul control din grup; fiecare control care urmează (în raport cu ordinea de selectare) și are proprietatea **Group** invalidată se află în același grup. Grupul se încheie la următorul control care are proprietatea **Group** validată. Deplasarea între controalele unui grup se face cu ajutorul tastelor săgeată.

Proiectul **CityBreak** folosește două grupuri de butoane de opțiune; primul dintre ele selectează un oraș destinație, iar cel de-al doilea selectează tipul de hotel dorit.

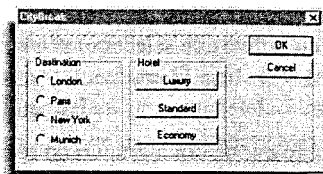
În cele ce urmează ar putea fi necesar să creșteți lățimea casetei de dialog și să modificați dimensiunile butoanelor.

1. Deschideți caseta de dialog `IDD_CITYBREAK_DIALOG` în cadrul editorului de resurse. Efectuați un clic pe eticheta **TODO: Place dialog controls here** și apoi apăsați tasta **Delete** pentru a o înlătura.
2. Selectați butonul **Group Box** de pe bara cu instrumente **Controls**; apoi adăugați o casetă de grupare în partea stângă a casetei de dialog, așa cum se vede în figura 4.4.
3. Asigurați-vă că noua casetă de grupare este selectată; apoi efectuați un clic cu butonul drept și selectați **Properties** din meniul contextual afișat. Este deschisă caseta de dialog **Group Box Properties**. Puteți menține această casetă de dialog deschisă efectuând un clic pe imaginea pionezei din colțul stânga sus.
4. Introduceți **Destination** în caseta **Caption** și validați opțiunea **Tab Stop**.
5. Selectați butonul **Radio Button** de pe bara cu instrumente **Controls** și adăugați patru butoane de opțiune în interiorul casetei de grupare.
6. Selectați butonul superior. Introduceți `IDC_LONDON` pentru **ID** și **London** pentru **Caption**, după care validați opțiunea **Group**.
7. Selectați cel de-al doilea buton. Introduceți `IDC_PARIS` pentru **ID** și **Paris** pentru **Caption**. Selectați cel de-al treilea buton. Introduceți `IDC_NEWYORK` pentru **ID** și **New York** pentru **Caption**. Selectați butonul inferior. Introduceți `IDC_MUNICH` pentru **ID** și **Munich** pentru **Caption**.

8. Selectați butonul Group Box de pe bara cu instrumente Controls; apoi creați o casetă de grupare în partea dreaptă a grupului **Destination**, așa cum se vede în figura 4.4.
9. Introduceți **Hotel** în caseta **Caption** și validați opțiunea **Tab Stop**.
10. Adăugați trei butoane de opțiune în interiorul cadrului **Hotel**.
11. Selectați toate cele trei butoane de opțiune pe care tocmai le-ați adăugat efectuând clic pe fiecare în timp ce țineți apăsată tasta Ctrl.
12. Efectuați un clic pe pagina **Styles** a casetei de dialog Properties. Validați opțiunea **Push-like**.
13. Selectați butonul superior. Introduceți IDC_LUXURY pentru **ID** și **Luxury** pentru **Caption**, după care validați opțiunea **Group**.
14. Selectați cel de-al doilea buton. Introduceți IDC_STANDARD pentru **ID** și **Standard** pentru **Caption**. Selectați cel de-al treilea buton. Introduceți IDC_ECONOMY pentru **ID** și **Economy** pentru **Caption**. Casetă de dialog ar trebui să arate acum ca în figura 4.4.

FIGURA 4.4

Aspectul casetei de dialog CityBreak.



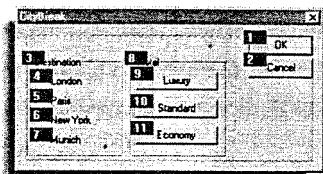
Ordinea de selectare este importantă pentru funcționarea corectă a butoanelor de opțiune, deoarece pe ea se bazează identificarea următorului control din grup. Pentru a inspecta ordinea de selectare, alegeți **Tab Order** din meniul **Layout**. Comparați cu ordinea de selectare ilustrată în figura 4.5. Pentru a verifica faptul că grupurile de butoane de opțiune au fost configurate corect, apăsați Ctrl+T sau selectați **Test** din meniul **Layout**. Ar trebui să puteți selecta exact unul dintre orașe și unul dintre tipurile de hotel. Pentru a încheia testarea apăsați tasta Esc.

VEDEȚI...

➤ Pentru mai multe informații despre proiectarea casetelor de dialog, vedeți pagina 60.

FIGURA 4.5

Ordinea de selectare din cadrul casetei de dialog CityBreak.



Identificarea butonului de opțiune selectat

Scopul unui grup de butoane de opțiune este să permită alegerea unui element din mai multe. La un moment dat, programul trebuie să identifice butonul selectat și să acționeze în consecință. În acest scop, unui grup de butoane de opțiune i se asociază o variabilă. Variabila

este asociată de fapt cu primul buton de opțiune din grup (cel care are validată opțiunea **Group**). În exemplul CityBreak, aceste butoane sunt IDC_LONDON și IDC_LUXURY.

Neprețuitul ClassWizard

ClassWizard este un instrument neprețuit, care își dovedește utilitatea pe tot parcursul dezvoltării unui proiect. ClassWizard vă ajută să creați și să gestionați clasele noi sau cele deja existente din proiect. Vă ajută, de asemenea, la întreținerea funcțiilor de tratare a mesajelor și la supradefinirea funcțiilor virtuale implementate în cadrul bibliotecii MFC.

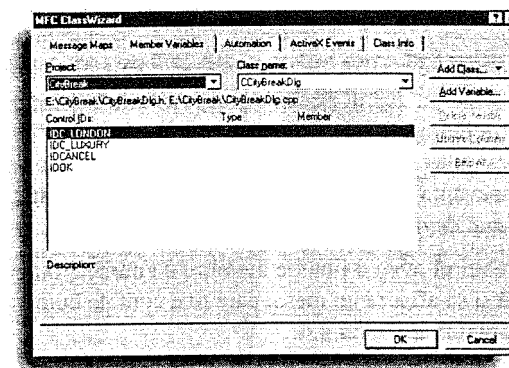
Adăugarea acestor variabile și asocierea lor cu butoanele de opțiune adecvate se fac prin intermediul instrumentului ClassWizard. În cazul exemplului CityBreak este necesară câte o variabilă pentru fiecare grup.

Asocierea variabilelor cu butoanele de opțiune prin intermediul ClassWizard

1. Apăsați Ctrl+W pentru a lansa ClassWizard sau selectați **ClassWizard** din meniul **View**. Este afișată caseta de dialog MFC ClassWizard, înfățișată în figura 4.6.
2. Selectați pagina **Member Variables**.
3. Selectați din caseta combinată **Class name**: clasa asociată dialogului; pentru exemplul nostru, selectați **CCityBreakDlg**.
4. Selectați din caseta cu listă **Control IDs**: ID-ul controlului; pentru exemplul nostru, selectați IDC_LONDON.
5. Efectuați un clic pe butonul **Add Variable**. Este afișată caseta de dialog Add Member Variable (vedeți figura 4.7). Rețineți că adăugarea unei variabile membru asociată unui control se poate face și prin efectuarea unui dublu clic pe ID-ul controlului.

FIGURA 4.6

Casetă de dialog ClassWizard.



6. Introduceți numele variabilei în caseta **Member Variable name**; pentru exemplul nostru, introduceți **m_nDestination**. Apoi efectuați un clic pe OK pentru a reveni în pagina **Member Variables** din ClassWizard.
7. Selectați IDC_LUXURY din caseta cu listă **Control IDs**; apoi efectuați un clic pe butonul **Add Variable** pentru a deschide dialogul Add Member Variable.

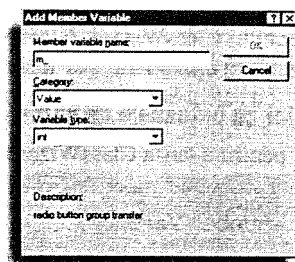
8. Introduceți `m_nHotel` în caseta **Member Variable Name**; apoi efectuați un clic pe **OK** pentru a adăuga noua variabilă membru.

9. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ClassWizard**.

Acum variabilele au fost adăugate, așa că trebuie inițializate. **ClassWizard** a generat în cadrul programului codul care inițializează valorile cu `-1`, ceea ce înseamnă că, dacă asamblați și rulați programul în acest moment, la afișarea casetei de dialog nu va fi selectat nici unul dintre butoanele de opțiune. Aceasta se dorește de puține ori; de regulă, inițial este selectat primul buton de opțiune din fiecare grup.

FIGURA 4.7

Caseta de dialog **Add Member**.



În acest scop va trebui să editați codul de inițializare generat de **ClassWizard** în constructorul clasei `CCityBreakDlg`. Selectați pagina **ClassView**, expandați clasa `CCityBreakDlg` și efectuați un dublu clic pe primul element subordonat. În fereastra editor este afișată funcția constructor. Editați cele două linii de cod pentru a inițializa variabilele cu zero, așa cum se vede aici.

```
m_nDestination = 0;
m_nHotel = 0;
```

Singurul lucru care rămâne de făcut este să adăugați cod care să identifice butoanele de opțiune selectate în cadrul fiecărui grup. Aceasta se face prin adăugarea unei funcții de tratare pentru butonul **OK** și afișarea unui mesaj **Bon Voyage**. Adăugați o funcție de tratare a mesajului `BN_CLICKED` corespunzător butonului `IDOK`. Pentru informații ajutătoare consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton”, prezentată mai devreme în acest capitol.

Acum ar trebui să aveți o funcție membru cu numele `OnOK()`. Pentru ca această funcție să pregătească și să afișeze un mesaj care ține cont de butoanele de opțiune selectate, adăugați codul prezentat în listingul 4.4.

LISTINGUL 4.4 LST04_4.CPP – Codul pentru identificarea butoanelor de opțiune selectate

```
1 void CCityBreakDlg::OnOK()
2 {
3     // TODO: Add extra validation here
4     CString strMessage;
```

```
5     CString strHotel;
6     CString strDest;
7
8     // ** Preluam in variabile informatiile despre controale
9     UpdateData(); — ❶
10
11    // ** Extragem eticheta fiecaruia dintre butoanele de optiune selectate
12    GetDlgItem(IDC_LUXURY + m_nHotel)->GetWindowText(strHotel); — ❷
13    GetDlgItem(IDC_LONDON + m_nDestination)->GetWindowText(strDest); — ❸
14
15    // ** Formataam si afisam mesajul
16    strMessage = "Bon Voyage to a " + strHotel + " Hotel in " — ❹
17    + strDest;
18    MessageBox(strMessage);
19
20    CDialog::OnOK();
21 }
```

- ❶ `UpdateData()` preia informații de la controalele din caseta de dialog și actualizează variabilele asociate acestora.
- ❷ Se preia eticheta butonului de opțiune selectat din grupul legat de tipul hotelului.
- ❸ Se preia eticheta butonului de opțiune selectat din grupul legat de destinație.
- ❹ Se compune un mesaj care conține etichetele butoanelor selectate.

Funcția `UpdateData()` apelată în linia 9 preia valoarea controalelor în variabilele asociate acestora. După linia 11, valorile `m_nDestination` și `m_nHotel` vor fi numere relative la primul control. De exemplu, dacă este selectat butonul **London**, `m_nDestination` va avea valoarea 0; dacă s-a selectat **Paris**, valoarea va fi 1. Aceste valori sunt utilizate în liniile 12 și 13 pentru a prelua etichetele butoanelor selectate. În linia 16 se folosește operatorul `+` (plus) din `CString` pentru a compune un mesaj în cadrul variabilei `strMessage`; acest mesaj este afișat de apelul funcției `MessageBox()` din linia 17.

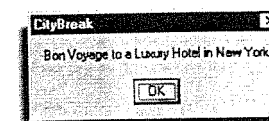
Acum asamblați și rulați programul. Testați codul încercând mai multe combinații de destinații și de tipuri de hoteluri. Dacă la efectuarea unui clic pe butonul **OK** erau selectate **New York** și **Luxury**, ar trebui să fie afișat mesajul ilustrat în figura 4.8.

VEDEȚI...

► Pentru a afla cum funcționează schimburile de date cu un dialog, consultați pagina 207.

FIGURA 4.8

Un mesaj **Bon Voyage**.



Utilizarea casetelor de validare

Casetele de validare sunt extrem de des întâlnite în aplicații; ați folosit deja multe dintre ele pentru a selecta diferitele proprietăți și stiluri ale controalelor în cadrul editorului de resurse. Casetele de validare permit utilizatorului să selecteze nici una, una sau mai multe opțiuni. O casetă de validare standard afișează o etichetă alături de o casetă mică. Caseta afișează un marcaj de bifare (sau o cruce, în Windows NT) dacă elementul este selectat și este goală atunci când elementul nu este selectat. O casetă de validare poate opera ca și control *tri-stare*, având o a treia stare în care marcajul de bifare (sau crucea) apare estompat. Așa ceva se folosește atunci când starea reprezentată de către control nu poate fi determinată. De exemplu, în cazul în care proiectați o casetă de dialog în cadrul editorului de resurse și selectați mai multe controale care au stiluri diferite, casetele de validare corespunzătoare stilurilor vor fi estompeate.

Crearea casetelor de validare

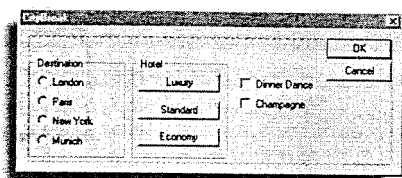
Pentru a studia modul de utilizare a casetelor de validare, proiectul CityBreak poate fi dezvoltat puțin. Vom adăuga două casete de validare pentru a permite turistului să decidă dacă dorește să participe la o serată dansantă pe timpul șederii și, în caz că da, dacă dorește șampanie.

Adăugarea casetelor de validare în cadrul casetei de dialog CityBreak

1. Deschideți caseta de dialog `IDD_CITYBREAK_DIALOG` în cadrul editorului de resurse.
2. Selectați butonul Check Box de pe bara cu instrumente Controls și adăugați două casete de validare, așa cum se vede în figura 4.9.
3. Selectați caseta de validare superioară; apoi efectuați un clic cu butonul drept și selectați **Properties** din meniul contextual afișat. Va apărea caseta de dialog Check Box Properties.
4. Introduceți un nume în caseta combinată **ID**; pentru exemplul nostru, introduceți `IDC_DANCE`. Introduceți o etichetă în caseta **Caption**; pentru exemplul nostru, introduceți Dinner Dance.
5. Selectați opțiunea **Group**. Aceasta marchează încheierea grupului de butoane de opțiune **Hotel**.
6. Selectați a doua casetă de validare.
7. Introduceți `IDC_CHAMPAGNE` în caseta combinată **ID** și Champagne în caseta **Caption**. Caseta de dialog ar trebui să arate acum ca în figura 4.9.

FIGURA 4.9

Caseta de dialog CityBreak
conținând casete de validare.



Citirea și modificarea stării casetelor de validare

Ca și în cazul butoanelor de opțiune, scopul casetelor de validare este să permită selectarea opțiunilor. La un moment dat, programul trebuie să afle starea casetelor de validare. În acest scop este utilizată `CButton`. Această clasă conține o funcție membru, `GetCheck()`, care obține starea casetei de validare, și o funcție `SetCheck()` pentru stabilirea stării.

Clasa CButton

`CButton` este clasa MFC care încapsulează funcționalitatea butoanelor de comandă, a butoanelor de opțiune, a casetelor de grupare și a casetelor de validare. Pentru a utiliza funcțiile membru ale acestei clase puteți să converțiți pointerul întors de `GetDlgItem()` la un pointer `CButton`. Aceste funcții vă permit să inspectați și să stabiliți starea butoanelor de opțiune și a casetelor de validare, precum și să inspectați și să stabiliți indicatorii de stil ai butoanelor.

Codul prezentat ca exemplu demonstrează modul în care pot fi puse în legătură două casete de validare. Nu puteți beneficia de șampanie dacă nu luați parte la serată, însă puteți participa la serată fără a bea șampanie. Din această cauză, dacă **Champagne** este selectat, **Dinner Dance** este selectat automat, iar dacă **Dinner Dance** este invalidat, **Champagne** este invalidat automat.

Adăugați funcții de tratare a mesajului `BN_CLICKED` pentru casetele de validare `IDC_DANCE` și `IDC_CHAMPAGNE`. Pentru informații ajutătoare consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton”, prezentată mai devreme în acest capitol.

Acum aveți scheletele funcțiilor, așa că puteți adăuga codul prezentat în listingul 4.5.

LISTINGUL 4.5 LST04_5.CPP – Accesarea stării casetelor de validare

```

1 void CCityBreakDlg::OnChampagne() — 1
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Obținem un pointer la fiecare dintre obiectele caseta de
    validare
6     CButton* pDance = (CButton*)GetDlgItem(IDC_DANCE); — 2
7     CButton* pChamp = (CButton*)GetDlgItem(IDC_CHAMPAGNE); — 3
8
9     // ** Dacă s-a optat pentru sampanie, selectam serata
10    if (pChamp->GetCheck()) — 4
11        pDance->SetCheck(1);
12 }
13
14 void CCityBreakDlg::OnDance() — 5
15 {
16     // TODO: Add your control notification handler code
17
```

(continuare)

LISTINGUL 4.5 Continuare

```

18 // ** Obținem un pointer la fiecare dintre obiectele caseta de
    validare
19 CButton* pDance = (CButton*)GetDlgItem(IDC_DANCE); — 6
20 CButton* pChamp = (CButton*)GetDlgItem(IDC_CHAMPAGNE); — 7
21
22 // ** Daca serata nu este selectata, deselectam sampania
23 if (!pDance->GetCheck()) — 8
24     pChamp->SetCheck(0);
25 }

```

- 1 OnChampagne() este funcția de tratare a mesajului apelată la efectuarea unui clic pe caseta de validare Champagne.
- 2 Se obține un pointer CButton la caseta de validare IDC_DANCE.
- 3 Se obține un pointer CButton la caseta de validare IDC_CHAMPAGNE.
- 4 În cazul în care caseta de validare Champagne este selectată, ne asigurăm că și caseta de validare Dance este selectată.
- 5 OnDance() este funcția de tratare a mesajului apelată la efectuarea unui clic pe caseta de validare Dance.
- 6 Se obține un pointer CButton la caseta de validare IDC_DANCE.
- 7 Se obține un pointer CButton la caseta de validare IDC_CHAMPAGNE.
- 8 În cazul în care caseta de validare Dance nu este selectată, ne asigurăm că nici caseta de validare Champagne nu este selectată.

Puteți observa în liniile 6, 7, 19 și 20 că pointerul `CWnd` întors de funcția `GetDlgItem()` este convertit la un pointer `CButton`. `CButton` este o clasă MFC care încapsulează funcționalitatea tuturor tipurilor de butoane. Motivul pentru care `CButton` nu a fost folosită în cazul butoanelor de comandă este faptul că nu a fost necesar; s-a putut utiliza direct pointerul la `CWnd`. Diferența dintre un buton de comandă și o casetă de validare constă în faptul că un buton de comandă nu are o stare. Conversia la un pointer `CButton` este necesară pentru a putea apela funcțiile membru ale acestei clase, `GetCheck()` și `SetCheck()`.

Instrucțiunea `if` din linia 10 identifică starea casetei de validare **Champagne** și, în cazul în care aceasta este selectată, apelează `SetCheck()` pentru caseta de validare **Dinner Dance**, transmițând valoarea 1 pentru a selecta caseta. Instrucțiunea `if` din linia 23 face exact contrariul, invalidând caseta **Champagne** în cazul în care **Dinner Dance** este invalidată.

Asamblați și rulați programul. Verificați că selectarea sau deselectarea controlului corespunzător stabilește automat starea celui alt control.

Pentru a finisa exemplul, dezvoltăți funcționalitatea rutinei `OnOK()` prin adăugarea codului dintre liniile 19 și 29 din listingul 4.6.

LISTINGUL 4.6 LST04_6.CPP – Adăugarea de informații suplimentare în mesajul Bon Voyage

```

1 void CCityBreakDlg::OnOK()
2 {
3     // TODO: Add extra validation here
4     CString strMessage;
5     CString strHotel;
6     CString strDest;
7
8     // ** Preluam in variabile informatiile despre controale
9     UpdateData();
10
11     // ** Extragem eticheta fiecaruia dintre butoanele de optiune
        ?selectate
12     GetDlgItem(IDC_LUXURY + m_nHotel)->GetWindowText(strHotel); — 1
13     GetDlgItem(IDC_LONDON + m_nDestination)->GetWindowText(strDest);
14                                     — 2
15
16     // ** Formataam si afisam mesajul
17     strMessage = "Bon Voyage to a " + strHotel + " Hotel in "
        + strDest;
18
19     // ** Obținem cate un pointer la fiecare caseta de validare
20     CButton* pDance = (CButton*)GetDlgItem(IDC_DANCE);
21     CButton* pChamp = (CButton*)GetDlgItem(IDC_CHAMPAGNE);
22
23     // ** Completam mesajul in concordanta cu starea casetelor de
        validare
24     if(pDance->GetCheck())
25     {
26         strMessage += " and a Dinner Dance";
27         if(pChamp->GetCheck())
28             strMessage += " with Champagne";
29     }
30     MessageBox(strMessage);
31
32     CDialog::OnOK();
33 }

```

- 1 Obținem eticheta butonului de opțiune selectat în cadrul grupului legat de tipul hotelului.
- 2 Obținem eticheta butonului de opțiune selectat în cadrul grupului legat de destinație.

Funcția `OnOK()` ține cont acum și de starea casetelor de validare și afișează informații suplimentare în cadrul mesajului **Bon Voyage**. În liniile 19 și 20, pointerul `CWnd` întors de către `GetDlgItem()` este convertit la un pointer `CButton`. După aceste linii, `pDance` este

un pointer la caseta de validare IDC_DANCE, iar pChamp este un pointer la caseta de validare IDC_CHAMPAGNE. Acești pointeri sunt apoi utilizați pentru a verifica starea fiecărui control, în liniile 24 și, respectiv, 27. În cazul în care controlul IDC_DANCE este validat, la variabila strMessage se atașează șirul and a Dinner Dance prin intermediul operatorului += oferit de CString. În cazul în care controlul IDC_CHAMPAGNE este selectat, strMessage este din nou modificat, fiindu-i atașat șirul with Champagne.

Dacă asamblați și rulați aplicația, la efectuarea unui clic pe butonul OK va fi afișat un mesaj conținând informații adiționale, în funcție de casetele de validare (vedeți figura 4.10).

FIGURA 4.10

Mesajul Bon Voyage extins.



CAPITOLUL

5

Utilizarea controalelor textuale

Afișarea de mesaje și informații textuale în casetele de dialog

Manipularea controalelor textuale în timpul execuției

Validarea datelor odată cu introducerea lor

Extinderea funcționalității controalelor prin intermediul subclasării

Timp de mulți ani de zile, înainte de apariția sistemului Windows și a interfeței grafice cu utilizatorul, majoritatea calculatoarelor erau limitate la afișarea a 24 de linii de text cu câte 80 de caractere fiecare. Predomina afișarea cu un singur tip de caractere, cu verde pe negru; cei mai nefericiți aveau terminale cu arămiu pe negru. Contrar celor susținute de unii specialiști, aceste terminale monocrome nu au dispărut încă. Motivul este faptul că marea majoritate a informațiilor introduse în calculatoare și apoi oferite nouă are încă un caracter textual, ceea ce permite ca astfel de sisteme să-și păstreze utilitatea. Cu sau fără interfață grafică cu utilizatorul, este încă necesar de multe ori ca aplicațiile să afișeze sau să primească de la utilizator informații textuale.

În cadrul casetelor de dialog, informațiile textuale sunt afișate cu ajutorul controalelor etichetă statică, iar textul se introduce prin intermediul controalelor casetă de editare.

Utilizarea controalelor etichetă statică

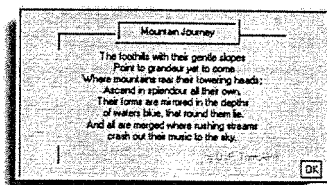
Dacă ați întreba pe cineva câte cuvinte valorează o imagine, replica ar fi probabil „o mie”, dar dacă ați arăta o aceeași imagine la două persoane și le-ați solicita fiecareia un singur cuvânt descriptiv, răspunsurile lor ar fi probabil diferite. Atât timp cât cuvintele scrise sunt prezentate într-o limbă cunoscută cititorului, ele reprezintă cea mai bună metodă de a comunica un sens precis. Astfel de informații textuale sunt necesare în mod uzual în casetele de dialog. Adeseori se adaugă text cu rol de etichetă pentru controale sau pentru a descrie o opțiune, sau uneori doar pentru a afișa o propoziție, așa cum ar fi un mesaj privind drepturile de autor.

Formatarea textului din casetele de dialog

Atunci când într-o casetă de dialog este nevoie de afișarea unui text simplu, apăsați la controlul etichetă statică. Fiecare control etichetă statică poate afișa până la 255 de caractere, introduse ca și etichetă. Puteți folosi caracterul linie nouă (\n) pentru a desfășura textul pe mai multe linii, fiind posibilă alinierea la stânga, la dreapta sau pe centru. O serie de opțiuni de stil permit încadrarea controlului cu margini ridicate, adâncite sau de alte diverse tipuri. Prin intermediul etichetelor statice și cu puțină ingeniozitate, casetele de dialog pot căpăta un aspect destul de elegant, așa cum o ilustrează figura 5.1.

FIGURA 5.1

Mountain Journey: un exemplu de controale etichetă statică.



Combinarea etichetelor statice și a casetelor de editare

Controalele etichetă statică sunt utilizate cel mai des în casetele de dialog pe post de etichete descriptive pentru casetele de editare. *Casetele de editare* sunt controale care permit utilizatorului să introducă informații. Controalele etichetă statică sunt plasate alături

de controalele casetă de editare ca etichete ale acestora din urmă, deoarece casetele de editare nu dispun de etichete proprii.

Puteți să plasați un control etichetă statică înaintea unei casete de editare pentru a crea o scurtătură la caseta de editare. Aceasta se face prin specificarea unei mnemonice în textul etichetei statice. O mnemonică oferă posibilitatea de accesare a unui control cu ajutorul etichetei statice. Tasta mnemonică se precizează în etichetă printr-un simbol ampersand (&) plasat imediat înaintea literei care va fi folosită în combinație. De exemplu, eticheta 'Fot&bal' specifică litera 'b' sau 'B' ca tastă mnemonică. Eticheta este afișată sub forma Fotbal, cu litera mnemonică subliniată. Fiecare mnemonică dintr-o casetă de dialog trebuie să fie unică; pentru a identifica dublurile puteți utiliza opțiunea **Check Mnemonics** a meniului de context din cadrul editorului de resurse.

Dezactivarea mнемonicelor

Dacă la tastarea unei combinații mnemonice controlul următor este dezactivat, focusul se transferă următorului control activat. Pentru a împiedica acest comportament este necesară subclasarea controlului etichetă statică.

Un control etichetă statică nu poate primi focusul, așa că la tastarea combinației mnemonice se transferă focusul următorului control din ordinea de selectare. La primirea focusului de către o casetă de editare, în interiorul acesteia este plasat automat un cursor, fiind posibilă introducerea datelor.

VEDEȚI...

➤ Pentru a afla cum se pot adăuga controale casetă de editare, vedeți pagina 100.

Modificarea controalelor etichetă statică la momentul execuției

Controalele etichetă statică se pot dovedi foarte utile atunci când tot ce doriți este să transmiteți un mesaj. Spre exemplu, o aplicație multi-utilizator ar putea avea o casetă de dialog de stare care afișează numărul curent de utilizatori activi. Controalele etichetă statică sunt ideale pentru astfel de situații. Evident, în acest caz numărul de utilizatori poate varia, așa că pe durata execuției programului va trebui modificat conținutul controlului etichetă. În secțiunea de față veți crea o casetă de dialog conținând informații despre sistem. Prin intermediul controalelor etichetă statică veți afișa numele calculatorului și al utilizatorului și cantitatea de memorie totală și disponibilă.

Mai întâi apăsați la AppWizard pentru a crea un nou proiect de tip dialog, numit Sysinfo.

Odată ce ați creat noul proiect, va trebui să adăugați în caseta de dialog Sysinfo opt controale etichetă statică și un control buton.

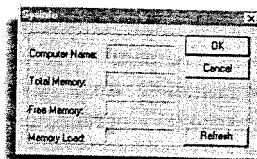
Adăugarea controalelor etichetă statică într-o casetă de dialog

1. Deschideți caseta de dialog `IDD_SYSINFO_DIALOG` în cadrul editorului de resurse. Efectuați un clic pe eticheta `TOD0: Place Dialog Controls Here` și apoi apăsați tasta Delete pentru a o înlătura.

2. Selectați butonul etichetei statice de pe bara cu instrumente Controls și creați patru controale etichetă în partea stângă a casetei de dialog, așa cum se vede în figura 5.2.
3. Modificați textul fiecărui control în concordanță cu figura 5.2.
4. Selectați butonul etichetei statice de pe bara cu instrumente Controls și adăugați un nou control în partea dreaptă a etichetei **Computer Name**.
5. Introduceți un nume în caseta combinată **ID**; pentru exemplul nostru, introduceți `IDC_COMPUTER_NAME`.
6. Selectați butonul etichetei statice de pe bara cu instrumente Controls și adăugați un control în partea dreaptă a etichetei **Total Memory**.
7. Introduceți `IDC_TOTAL_MEMORY` în caseta combinată **ID**.
8. Selectați butonul etichetei statice de pe bara cu instrumente Controls și adăugați un control în partea dreaptă a etichetei **Free Memory**.
9. Introduceți `IDC_FREE_MEMORY` în caseta combinată **ID**.
10. Selectați butonul etichetei statice de pe bara cu instrumente Controls și adăugați un control în partea dreaptă a etichetei **Memory Load**.
11. Introduceți `IDC_MEMORY_LOAD` în caseta combinată **ID**.
12. Țineți apăsată tasta Ctrl și selectați toate cele patru controale etichetă pe care tocmai le-ați adăugat. Ștergeți cuvântul Static din caseta de editare a etichetei, după care selectați pagina **Styles** și validați opțiunea **Sunken**.
13. Selectați pictograma de buton de pe bara cu instrumente Controls și adăugați un buton în partea dreaptă jos a casetei de dialog.
14. Introduceți `IDC_REFRESH` în caseta combinată **ID** și specificați Refresh ca și etichetă. Caseta de dialog ar trebui să arate acum ca în figura 5.2.

FIGURA 5.2

Aspectul casetei de dialog Sysinfo.



Cele patru controale cu aspect adâncit vor fi actualizate pentru a afișa informațiile corespunzătoare de fiecare dată când se efectuează clic asupra butonului **Refresh**. În acest scop, asociați câte o variabilă cu fiecare dintre controale prin intermediul pașilor de mai jos.

IDC_STATIC este un identificator aparte

În mod implicit, controalele etichetă statică primesc același ID, `IDC_STATIC`. Acesta este un identificator aparte care are valoarea -1. Pentru asocierea unei variabile cu un control etichetă statică este necesară înlocuirea acestui ID implicit.

Asocierea variabilelor cu controalele etichetă prin intermediul ClassWizard

1. Apăsăți Ctrl+W pentru a lansa ClassWizard sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Member Variables**.
3. Selectați din caseta combinată **Class Name**: clasa asociată dialogului; pentru exemplul nostru, selectați `CSysinfoDlg`.
4. Selectați din caseta cu listă **Control IDs** identificatorul controlului etichetă; pentru exemplul nostru selectați `IDC_COMPUTER_NAME`. Apoi efectuați un clic pe butonul **Add Variable**; va fi afișată caseta de dialog Add Member Variable. Rețineți că adăugarea unei variabile membru asociată unui control se poate face și prin efectuarea unui dublu clic pe identificatorul controlului.
5. Asigurați-vă că în caseta combinată **Category** este selectat **Value** și că în caseta **Variable Type** este selectat **cstring**.
6. Introduceți numele variabilei în caseta **Member Variable Name**; pentru exemplul nostru, introduceți `m_strComputerName`. Efectuați un clic pe **OK**.
7. Selectați `IDC_TOTAL_MEMORY` din lista **Control IDs** și apoi efectuați un clic pe butonul **Add Variable**.
8. Introduceți `m_strTotalMemory` în caseta **Member Variable Name** și apoi efectuați un clic pe **OK**.
9. Selectați `IDC_FREE_MEMORY` din lista **Control IDs** și apoi efectuați un clic pe butonul **Add Variable**.
10. Introduceți `m_strFreeMemory` în caseta **Member Variable Name** și apoi efectuați un clic pe **OK**.
11. Selectați `IDC_MEMORY_LOAD` din lista **Control IDs** și apoi efectuați un clic pe butonul **Add Variable**.
12. Introduceți `m_strMemoryLoad` în caseta **Member Variable Name** și apoi efectuați un clic pe **OK**.
13. Efectuați un clic pe **OK**.

Acum adăugați o funcție de tratare a mesajului `BN_CLICKED` pentru butonul `IDC_REFRESH`. Pentru informații ajutătoare privind adăugarea funcțiilor de tratare a mesajelor, consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton” prezentată în capitolul 4, „Utilizarea controalelor buton”. Rezultatul ar trebui să fie crearea unei funcții membru numite `OnRefresh()`. Pentru ca această funcție să actualizeze variabila `cstring` asociată cu fiecare control etichetă în scopul afișării informațiilor corecte, adăugați codul din listingul 5.1.

LISTINGUL 5.1 LST05_1.CPP – Modificarea controalelor etichetă statică la momentul execuției

```
1 void CSysinfoDlg::OnRefresh()
2 {
```

(continuare)

LISTINGUL 5.1 Continuare

```

3 // TODO: Add your control notification handler code here
4 // ** Variabile folosite pentru a obține numele calculatorului
5 TCHAR szBuffer[256];
6 DWORD dwSize = 256;
7
8 // ** Obținem numele calculatorului din Windows
9 GetComputerName(szBuffer, &dwSize); — ①
10
11 // ** Plasăm numele calculatorului în variabila
12 // ** membru corespunzătoare
13 m_strComputerName = szBuffer;
14
15 // ** Alocăm o structură pentru a prelua informații despre memorie
16 MEMORYSTATUS mem_stat; — ②
17
18 // ** Obținem starea curentă a memoriei
19 GlobalMemoryStatus(&mem_stat);
20
21 // ** Plasăm informațiile despre memorie în
22 // ** variabilele membru corespunzătoare
23 m_strTotalMemory.Format("%ld KB", — ③
24                             mem_stat.dwTotalPhys / 1024);
25 m_strFreeMemory.Format("%ld KB",
26                          mem_stat.dwAvailPhys / 1024);
27 m_strMemoryLoad.Format("%d %%",
28                          mem_stat.dwMemoryLoad);
29
30 // ** Actualizăm conținutul afisat al controalelor
31 UpdateData(FALSE);
32 }

```

- ① Este obținut numele calculatorului și plasat într-o variabilă șir de caractere.
- ② Structura MEMORYSTATUS conține informații despre memoria fizică și virtuală.
- ③ Variabilele șir de caractere se folosesc pentru a formata informațiile despre memorie și a le afișa în cadrul casetei de dialog.

În liniile 5 și 6 se declară două variabile locale care sunt transmise apoi în apelul funcției globale `GetComputerName()` din linia 9. Parametrii acestei funcții sunt un buffer de caractere, în care funcția plasează numele calculatorului, și adresa unei variabile `DWORD` care reține lungimea maximă a bufferului de caractere. Variabila buffer de caractere se declară cu ajutorul macro-ului `TCHAR` (în linia 5). Acest macro este utilizat cu scopul de a face programul compatibil cu ambele seturi de caractere, ANSI și Unicode. Setul de caractere Unicode reprezintă fiecare caracter pe doi octeți și este un standard dezvoltat pentru a codifica toate simbolurile din lume.

Funcții sistem utile

Pe lângă `GetComputerName()`, puteți de asemenea să apelați `GetUserName()` pentru a afla cine rulează o aplicație sau `GetVersionEx()` pentru a identifica sistemul de operare pe care aceasta rulează.

În linia 13, conținutul bufferului este transferat în variabila `m_strComputerName`, care este asociată cu controlul etichetă `IDC_COMPUTER_NAME`.

În linia 16, variabila `mem_stat` este declarată ca structură de tip `MEMORYSTATUS`. Membrii acestei structuri sunt prezentați în listingul 5.2. În linia 19, adresa variabilei `mem_stat` este transmisă în apelul funcției globale `GlobalMemoryStatus()`, care înscrie în elementele structurii informații despre starea curentă a memoriei. Funcția `CString::Format()` este utilizată în liniile 23, 25 și 27 pentru prelucrarea informațiilor din elementele structurii `mem_stat` și actualizarea variabilelor `CString` asociate cu fiecare dintre controalele etichetă corespunzătoare detaliilor despre memorie.

În apelul `UpdateData()` din linia 31 se transmite parametrul `FALSE` pentru a actualiza afișarea celor patru controale etichetă statică.

După asamblarea și rularea proiectului `Sysinfo`, de fiecare dată când efectuați un clic pe butonul **Refresh** vor fi actualizate informațiile despre memorie. Încercați să deschideți și să închideți alte aplicații, după care efectuați clic pe **Refresh**. Caseta de dialog ar trebui să arate ca în figura 5.3. S-ar putea să vă întrebați de ce valoarea **Memory Load** (procentul de memorie folosită) nu pare a corespunde; aceasta se întâmplă deoarece se ia în calcul și *memoria virtuală*, care este de fapt un fișier folosit pe post de memorie.

Memoria virtuală

Pentru ca aplicațiile Windows să poată folosi mai multă memorie decât cea fizică prezentă, o zonă de pe hard-disc joacă rolul de memorie. Această memorie poartă numele de memorie virtuală.

LISTINGUL 5.2 LST05_2.CPP – Structura MEMORYSTATUS

```

1 typedef struct _MEMORYSTATUS {
2     DWORD dwLength;
3     DWORD dwMemoryLoad;
4     DWORD dwTotalPhys;
5     DWORD dwAvailPhys;
6
7     DWORD dwAvailPageFile;
8     DWORD dwTotalVirtual;
9     DWORD dwAvailVirtual;
10 } MEMORYSTATUS;

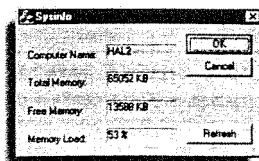
```

VEDEȚI...

- Pentru detalii privind crearea unei aplicații de tip dialog, vedeți pagina 54.
- Pentru mai multe informații despre proiectarea casetelor de dialog, vedeți pagina 60.
- Pentru informații ajutoare privind funcțiile de tratare, vedeți pagina 72.

FIGURA 5.3

Un exemplu cu afișarea informațiilor despre sistem.



Utilizarea controalelor casetă de editare

V-ați întrebat vreodată câte caractere de text sunt introduse în programele de calculator din toată lumea în fiecare zi – câți oameni introduc datele unei tranzacții bancare, ale unei comenzi sau cantitățile dintr-un inventar? Ei bine, oricâte caractere ar fi, se poate spune cu destulă certitudine că în Windows majoritatea acestora sosesc într-o casetă de editare.

Controalele casetă de editare sunt folosite pentru a recepționa textul introdus și pentru a afișa text. Într-o casetă de editare pot fi introduse și afișate caractere, numere și simbolurile uzuale prezente pe taste (inclusiv caractere cu accent). Există și anumite posibilități legate de editare. Spre exemplu, este posibilă selectarea cu mouse-ul sau cu tastatura a unui fragment sau a întregului text, acesta putând fi apoi decupat sau copiat și apoi inserat într-o altă casetă de editare. Prin efectuarea unui clic cu butonul drept se poate apela un meniu contextual care oferă următoarele opțiuni: **Undo** (anulare), **Cut** (decupare), **Copy** (copiere), **Paste** (lipire), **Delete** (ștergere) și **Select All** (selectarea întregului text).

Limitele casetelor de editare cu o singură linie

Casetele de editare cu o singură linie sunt limitate la maximum 32 KB, adică aproximativ 32000 de caractere.

Sunt disponibile mai multe opțiuni de stil pentru casetele de editare. Stilul **Password** maschează textul introdus de utilizator, de regulă prin înlocuirea caracterelor introduse cu un asterisc (*). Stilurile **Uppercase** și **Lowercase** transformă automat fiecare caracter tastat în majusculă sau, respectiv, minusculă. Stilul **Number** împiedică introducerea altor caractere decât cele numerice; rețineți că acest stil va împiedica și introducerea caracterului punct zecimal (.).

Următoarele secțiuni prezintă modul de utilizare a casetelor de editare, însă mai întâi trebuie să creați un nou proiect. Apelați la AppWizard pentru a crea un nou proiect de tip dialog cu numele Edits.

Acum ar trebui să aveți deschis în Developer Studio un proiect nou de tip dialog.

VEDEȚI...

➤ Pentru detalii privind modul de creare a unei aplicații de tip dialog, vedeți pagina 54.

Adăugarea casetelor de editare

Ca și în cazul tuturor celorlalte controale, casetele de editare se adaugă în cadrul machetelor de casetă de dialog prin intermediul editorului de resurse. Urmăți secvența de

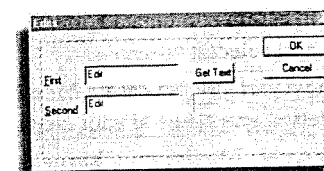
pași intitulată „Adăugarea casetelor de editare într-o casetă de dialog” pentru a modifica aspectul casetei de dialog Edits în concordanță cu figura 5.4.

Adăugarea casetelor de editare într-o casetă de dialog

1. Deschideți caseta de dialog `IDC_EDIT_DIALOG` în cadrul editorului de resurse. Efectuați un clic pe eticheta `TODO: Place Dialog Controls Here` și apoi apăsați tasta **Delete** pentru a o înlătura.
2. Selectați pictograma etichetei statice de pe bara cu instrumente **Controls**, după care adăugați un control etichetă în partea superioară stânga a casetei de dialog, așa cum se vede în figura 5.4.
3. Asigurați-vă că este selectat controlul etichetă, după care efectuați un clic cu butonul drept și selectați **Properties** din meniul de context afișat. Va apărea caseta de dialog **Text Properties**. Puteți menține această casetă de dialog vizibilă efectuând un clic pe imaginea pionezei din colțul stânga sus.
4. Introduceți o etichetă în caseta **Caption**; pentru exemplul nostru, introduceți `&First`.
5. Selectați pictograma casetei de editare de pe bara cu instrumente **Controls**, după care adăugați o casetă de editare în partea dreaptă a controlului etichetă **First**.
6. Introduceți un nume în caseta combinată **ID**; pentru exemplul nostru, introduceți `IDC_EDIT_FIRST`.
7. Selectați pictograma butonului de pe bara cu instrumente **Controls**, după care adăugați un buton în partea dreaptă a casetei de editare, așa cum se vede în figura 5.4.
8. Introduceți `IDC_GET_TEXT` în caseta **ID** și introduceți `Get Text` ca și etichetă.
9. Selectați pictograma etichetei statice de pe bara cu instrumente **Controls**, după care adăugați un control etichetă sub eticheta **First**.
10. Introduceți `&Second` în caseta **Caption**.
11. Selectați pictograma casetei de editare de pe bara cu instrumente **Controls**, după care adăugați o casetă de editare în dreapta controlului etichetă **Second**.
12. Selectați pictograma casetei de editare de pe bara cu instrumente **Controls**, după care adăugați o nouă casetă de editare în dreapta celei pe care tocmai ați adăugat-o.
13. Introduceți `IDC_EDIT_SHOW` în caseta **ID** și apoi selectați pagina **Styles** și validați opțiunea **Read-only**.
14. Măriți lungimea controlului pe care tocmai l-ați adăugat până la marginea din dreapta a casetei de dialog. Casetă de dialog ar trebui să arate ca în figura 5.4.

FIGURA 5.4

Aspectul casetei de dialog Edits.



Caseta de dialog Edits ilustrează modul în care mnemonicele din controalele etichetă statică au rolul de combinații de acces pentru casetele de editare. Puteți verifica acest lucru apăsând Ctrl+T sau selectând **Test** din meniul **Layout** pentru a afișa caseta de dialog în modul de testare. Odată ajuns în acest mod, combinația de taste Alt+F ar trebui să plaseze un cursor în interiorul controlului de editare din dreapta etichetei **First**. Combinația Alt+S ar trebui să activeze caseta de editare alăturată etichetei **Second**.

Scurtături către casetele de editare

Atunci când folosiți o mnemonică cu scopul de a crea o scurtătură către o casetă de editare, aceasta din urmă trebuie să fie succesoarea imediată a controlului etichetă statică în raport cu ordinea de selectare.

VEDEȚI...

➤ Pentru detalii privind crearea unei aplicații de tip dialog, vedeți pagina 23.

Modificarea și extragerea textului din casetele de editare

În multe situații în care casetele de dialog conțin casete de editare, textul introdus în acestea din urmă este accesat doar atunci când trebuie, de regulă când utilizatorul efectuează clic pe un buton așa cum este butonul **OK**. În acest moment, textul este preluat din control și stocat intern de către program sau, eventual, plasat într-un câmp al unei baze de date. Pentru asocierea unei variabile `CString` cu un control de editare puteți apela la `ClassWizard`. Acesta generează declarația variabilei și apelează macro-urile care se ocupă de schimbul de date dintre variabilă și control, permițându-vă să preluați și să actualizați cu ușurință textul dintr-o casetă de editare.

Asociererea unei variabile `CString` cu o casetă de editare prin intermediul `ClassWizard`

1. Apăsați Ctrl+W pentru a lansa `ClassWizard`, sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Member Variables**.
3. Selectați din caseta combinată **Class Name** clasa asociată dialogului; pentru exemplul nostru, selectați `CEditsDlg`.
4. Selectați din caseta cu listă **Control IDs** identificatorul controlului; pentru exemplul nostru, selectați `IDC_EDIT_FIRST`.
5. Efectuați un clic pe butonul **Add Variable**. Este afișată caseta de dialog **Add Member Variable**. Rețineți că adăugarea unei variabile membru asociată unui control se poate face și prin efectuarea unui dublu clic pe identificatorul controlului.
6. Asigurați-vă că în caseta combinată **Category** este selectat **Value** și că în caseta **Variable Type** este selectat `CString`.
7. Introduceți numele variabilei în caseta **Member Variable Name**; pentru exemplul nostru, introduceți `m_strFirst` și apoi efectuați un clic pe **OK**.
8. Introduceți 15 în caseta **Maximum Characters** din partea inferioară a casetei de dialog `ClassWizard`. Aceasta limitează lungimea șirului acceptat de către caseta de editare.
9. Efectuați un clic pe **OK**.

Acum, având controalele adăugate, vom utiliza butonul **Get Text** pentru extragerea textului din controlul de editare **First**, inversarea textului și înscrierea sa înapoi în controlul de editare. În acest scop, adăugați o funcție de tratare a mesajului `BN_CLICKED` pentru butonul `IDC_GET_TEXT`. Pentru informații ajutătoare privind funcțiile de tratare, consultați capitolul 4.

Acum ar trebui să aveți o funcție membru numită `OnGetText()`. Editați această funcție conform cu listingul 5.3.

LISTINGUL 5.3 LST05_3.CPP – Extragerea și modificarea textului dintr-o casetă de editare

```
1 void CEditsDlg::OnGetText()
2 {
3     // TODO: Add your control notification handler code here
4     // ** Transferam date de la control la variabila
5     UpdateData();
6
7     // ** Verificam daca s-a introdus un text
8     if (m_strFirst.IsEmpty() == FALSE) — ❶
9     {
10        // ** Afisam textul introdus
11        MessageBox(m_strFirst);
12
13        // ** Inversam caracterele din sir
14        m_strFirst.MakeReverse(); — ❷
15
16        // ** Transferam datele de la variabila
17        // ** catre control
18        UpdateData(FALSE); — ❸
19    }
20 }
```

❶ Ne asigurăm că șirul nu este vid.

❷ Ordinea caracterelor din variabila șir este inversată.

❸ Șirul inversat este afișat în cadrul casetei de dialog.

Apelul funcției `UpdateData()` din linia 5 folosește parametrul implicit `TRUE`, ceea ce are ca efect transferul textului de la control către variabila `m_strFirst`. În linia 8, funcția `IsEmpty()` din `CString` întoarce `FALSE` în cazul în care `m_strFirst` nu este vid. În linia 13, funcția membru `MakeReverse()` din `CString` inversează pur și simplu caracterele din șir, după care se apelează din nou `UpdateData()`, transmitând de această dată valoarea `FALSE` pentru a transfera conținutul variabilei acum modificate înapoi către control.

Tratarea mesajelor de informare asupra editării

Ceea ce am discutat până acum privește extragerea textului după introducerea acestuia. Dacă tratați mesajele trimise de fiecare dată când textul se modifică, este posibil să preluați textul pe măsură ce este introdus. Windows trimite opt mesaje de informare legate de controalele de editare. Aceste mesaje sunt prezentate în tabelul 5.1. De pildă, mesajul de informare `EN_CHANGE` este trimis de fiecare dată când textul din cadrul unui control se modifică.

TABELUL 5.1 Mesajele de informare asupra editării

Mesaj	Descriere
<code>EN_CHANGE</code>	Trimis pentru a informa că textul s-a modificat după afișarea acestuia.
<code>EN_UPDATE</code>	Trimis pentru a informa că textul s-a modificat înainte de afișarea acestuia.
<code>EN_SETFOCUS</code>	Trimis atunci când controlul primește focusul.
<code>EN_KILLFOCUS</code>	Trimis atunci când controlul pierde focusul.
<code>EN_MAXTEXT</code>	Trimis pentru a informa că textul a depășit lungimea permisă și a fost trunchiat.
<code>EN_HSCROLL</code>	Trimis atunci când se efectuează clic pe bara de derulare orizontală (numai pentru casetele multi-linie).
<code>EN_VSCROLL</code>	Trimis atunci când se efectuează clic pe bara de derulare verticală (numai pentru casetele multi-linie).
<code>EN_ERRSPACE</code>	Trimis în situația în care controlul nu poate alocă memorie.

Acum veți crea o funcție pentru tratarea mesajului `EN_CHANGE` trimis de caseta de editare `Second`. În cadrul acestei funcții veți extrage textul introdus curent și-l veți transfera în caseta de editare protejată la scriere. Astfel se va crea impresia că textul este introdus simultan în două locuri. Pentru adăugarea funcției de tratare veți apela la ClassWizard.

Până acum, funcțiile de tratare a mesajelor au fost adăugate prin intermediul casetei de dialog New Windows Message and Event Handler, dar, asemeni oricărei aplicații Windows care se respectă, Developer Studio permite efectuarea unei aceleiași operații în mai multe feluri. ClassWizard se poate dovedi mai util deoarece nu este limitat la operarea cu mesaje și elimină necesitatea de a deschide machetele casetelor de dialog.

Crearea unei funcții pentru tratarea mesajelor prin intermediul ClassWizard

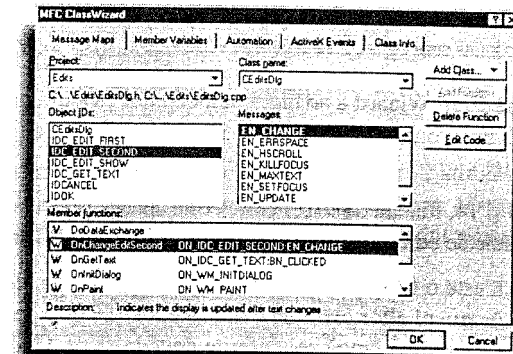
1. Apăsați `Ctrl+W` pentru a deschide ClassWizard sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Message Maps**.
3. Selectați clasa dialogului în cadrul casetei combinate **Class Name**; în exemplul nostru este vorba despre clasa `CEditDlg`.

4. Selectați identificatorul controlului în cadrul casetei cu listă **Object IDs**; pentru exemplul nostru selectați `IDC_EDIT_SECOND`.
5. Selectați mesajul dorit din caseta cu listă **Messages**; pentru exemplul nostru selectați `EN_CHANGE`.
6. Efectuați un clic pe butonul **Add Function**.
7. Efectuați un clic pe **OK** în cadrul casetei de dialog Add Member Function care prezintă `OnChangeEditSecond` ca fiind noua funcție membru. Caseta de dialog ClassWizard ar trebui să arate acum ca în figura 5.5.
8. Efectuați un clic pe **Edit Code** pentru a trece la editarea noii funcții de tratare.

Acum ar trebui să aveți la dispoziție scheletul funcției `OnChangeEditSecond()`; adăugați codul prezentat în listingul 5.4. Această funcție va fi apelată de fiecare dată când utilizatorul modifică textul aflat în cadrul controlului `IDC_EDIT_SECOND`, adică la ștergerea sau inserarea de caractere.

FIGURA 5.5

Caseta de dialog ClassWizard după adăugarea funcției de tratare a mesajului `EN_CHANGE`.



LISTINGUL 5.4 LST05_4.CPP – Interceptarea mesajului `EN_CHANGE`

```

1 void CEditDlg::OnChangeEditSecond()
2 {
3     // TODO: If this is a RICHEDIT control,
4     // the control will not send this notification
5     // unless you override the CDialog::OnInitDialog()
6     // function to send the EM_SETEVENMASK message
7     // to the control with the ENM_CHANGE flagORED
8     // into the lParam mask.
9
10    // TODO: Add your control notification handler here
11
12    // ** Obținem un pointer spre fiecare din controalele de editare
13    CEdit* pEdit = (CEdit*)GetDlgItem(IDC_EDIT_SECOND); ①

```

(continuare)

LISTINGUL 5.4 Continuare

```

14 CEdit* pEditShow = (CEdit*)GetDlgItem(IDC_EDIT_SHOW);
15
16 // ** Declaram o variabila care va retine textul introdus
17 CString strText;
18
19 // ** Extragem textul din caseta de editare "Second"
20 pEdit->GetWindowText(strText); — ❷
21
22 // ** Actualizam continutul casetei de editare protejate la scriere
23 pEditShow->SetWindowText(strText); — ❸
24 }

```

- ❶ Stabilim pointeri către fiecare dintre controalele de editare.
- ❷ Preluăm conținutul primului control de editare într-o variabilă șir.
- ❸ Facem conținutul celui de-al doilea control de editare identic cu cel al primului control.

Observați că ClassWizard a adăugat la începutul funcției o serie de comentarii adiționale privind utilizarea controalelor de editare formatată. Dar cum noi nu folosim un astfel de control, putem ignora comentariile fără nici o problemă.

În liniile 13 și 14, funcția `GetDlgItem()` este apelată pentru a obține câte un pointer la fiecare din controalele de editare cu care operăm. Acești pointeri sunt convertiți la tipul `CEdit`.

Clasa `CEdit` este o clasă din *biblioteca MFC* care încapsulează funcționalitatea casetelor de editare. Asemeni altor clase de controale, `CEdit` este derivată din `CWnd`. Clasa `CEdit` aduce multe funcții pe lângă cele oferite deja de `CWnd`. Unele dintre cele mai utile funcții sunt prezentate în tabelul 5.2.

Apelul `GetWindowText()` din linia 20 extrage textul aflat curent în caseta de editare și actualizează corespunzător variabila locală `strText`. Această variabilă este folosită apoi în linia 23 pentru a plasa textul extras în cadrul casetei de editare protejate la scriere.

Asamblați și rulați programul. Acum, ceea ce tastați în caseta de editare **Second** apare concomitent în controlul protejat la scriere aflat în dreapta. Puteți observa că la efectuarea unui clic pe suprafața controlului protejat la scriere este afișat un cursor, însă nu vă este permisă introducerea de text. Acest comportament poate fi modificat prin adăugarea unei funcții de tratare a mesajului `EN_SETFOCUS` pentru controlul `IDC_EDIT_SHOW`. Folosiți în ajutor pașii prezentați sub titlul „Crearea unei funcții pentru tratarea mesajelor prin intermediul ClassWizard”. Apoi adăugați următoarea linie de cod în cadrul funcției `OnSetFocusEditShow()` pe care ați creat-o:

```
GetDlgItem(IDC_EDIT_SECOND)->SetFocus();
```

Asamblați și rulați. Dacă efectuați acum un clic pe suprafața controlului protejat la scriere, cursorul va apărea în cadrul controlului de editare **Second**.

TABELUL 5.2 Funcții membru ale clasei `CEdit`

Numele funcției	Descriere
<code>Create()</code>	Permite crearea controalelor de editare la momentul execuției.
<code>GetSel()</code>	Furnizează poziția caracterelor de început și de sfârșit ale textului selectat curent.
<code>SetSel()</code>	Selectează o secvență de caractere.
<code>ReplaceSel()</code>	Înlocuiește textul selectat curent.
<code>LimitText()</code>	Stabilește numărul maxim permis de caractere.

VEDEȚI...

- Pentru informații ajutătoare privind adăugarea funcțiilor de tratare a mesajelor, vedeți pagina 72.
- Pentru detalii legate de transferul și validarea datelor din cadrul controalelor, vedeți pagina 207.
- Pentru alte amănunte privind clasa `CWnd`, vedeți pagina 72.

Subclasarea controalelor de editare

De multe ori textul trebuie introdus sub o anumită formă și de o anumită manieră. Spre exemplu, toate țările au coduri poștale; unele au coduri exclusiv numerice, în timp ce altele operează cu combinații de litere și cifre. Aplicațiile Windows care permit introducerea unor astfel de coduri ar trebui să le valideze după regulile corespunzătoare. Momentul cel mai potrivit pentru o astfel de validare este momentul în care codul este introdus. În acest scop este necesară o extindere a funcționalității casetei de editare prin subclasarea clasei `CEdit`. Subclasarea înseamnă crearea unei noi clase prin moștenirea atributelor și funcțiilor membru ale unei clase deja existente. Noua clasă poate folosi comportamentul clasei sale de bază, redefinind funcționalitatea acesteia acolo unde este necesar.

Ce este subclasarea?

O subclasă este o clasă derivată dintr-o clasă deja existentă și ea moștenește atributele și funcțiile membru ale clasei existente.

În această secțiune veți crea o nouă clasă care va fi derivată din `CEdit` și se va numi `CInitials`. Această nouă clasă va fi utilizată pentru a determina introducerea inițialelor de nume într-o anumită formă. Spre exemplu, singurele caractere care vor putea fi introduse sunt literele alfabetului și acestea vor fi transformate automat în majuscule. De asemenea, clasa va insera automat câte un punct după fiecare inițială și va șterge automat punctul corespunzător în momentul în care utilizatorul apasă tasta `Backspace`.

În acest scop va trebui mai întâi să adăugați alte câteva controale de editare în caseta de dialog `Edits`.

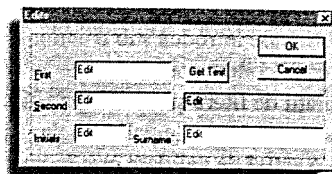
Adăugarea de noi casete de editare în caseta de dialog `Edits`

1. Deschideți caseta de dialog `IDD_EDITS_DIALOG` în cadrul editorului de resurse.

2. Selectați pictograma etichetei statice de pe bara cu instrumente Controls, după care adăugați un control etichetă în partea inferioară stânga a casetei de dialog, așa cum se vede în figura 5.6.
3. Introduceți `Initials` în caseta **Caption**.
4. Selectați pictograma casetei de editare de pe bara cu instrumente Controls, după care adăugați o casetă de editare în partea dreaptă a controlului etichetă **Initials**.
5. Introduceți `IDC_EDIT_INITIALS` în caseta combinată **ID**.
6. Selectați pictograma etichetei statice de pe bara cu instrumente Controls, după care adăugați un control etichetă în partea dreaptă a casetei de editare pe care tocmai ați creat-o (vedeți figura 5.6).
7. Introduceți `Surname` în caseta **Caption**.
8. Selectați pictograma casetei de editare de pe bara cu instrumente Controls, după care adăugați o casetă de editare în partea dreaptă a controlului etichetă **Surname**.
9. Extindeți lungimea noului control până în marginea din dreapta a casetei de dialog. Casetă de dialog ar trebui să arate acum ca în figura 5.6.

FIGURA 5.6

Casetă de dialog `Edits` extinsă.



După adăugarea casetelor de editare, creați clasa `CInitials`. Pe parcursul pașilor de mai jos veți putea observa că `ClassWizard` poate fi utilizat pentru a subclasa multe alte clase din *biblioteca MFC*, nu doar `CEdit`.

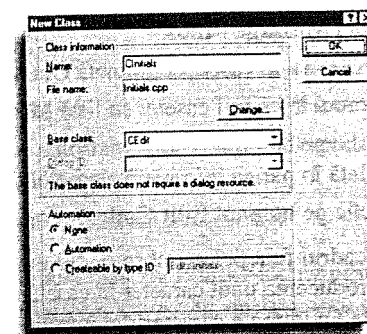
Crearea unei noi subclase prin intermediul `ClassWizard`

1. Apăsăți `Ctrl+W` pentru a deschide `ClassWizard` sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Message Maps**.
3. Selectați `CEdit` din caseta combinată **Class Name**.
4. Efectuați un clic pe butonul **Add Class** și apoi selectați **New** din meniul care apare. Va fi afișată caseta de dialog `New Class`, ilustrată în figura 5.7.
5. Introduceți `CInitials` în caseta **Name**.
6. Selectați `CEdit` în cadrul casetei combinate **Base Class**.
7. Efectuați un clic pe **OK** pentru a crea noua clasă.
8. Selectați `CEdit` din caseta combinată **Class Name**.
9. Selectați `IDC_EDIT_INITIALS` în cadrul casetei cu listă **Object IDs**.
10. Efectuați un clic pe butonul **Add Variable**. Va fi afișată caseta de dialog **Add Member Variable**.

11. Introduceți `m_edtInitials` în cadrul câmpului **Member Variable Name**.
12. Selectați **Control** din caseta combinată **Category**.
13. Selectați `CInitials` în caseta combinată **Variable Type**. În partea inferioară a casetei de dialog ar trebui să fie afișat mesajul **Map to CInitials Member (User-Defined-Class)**.
14. Efectuați un clic pe **OK** pentru a adăuga noua variabilă membru. Va fi afișat un mesaj care vă informează că este necesară adăugarea unei directive `#include`. Efectuați clic pe **OK**.
15. Efectuați un clic pe **OK** pentru a închide caseta de dialog `ClassWizard`.

FIGURA 5.7

Casetă de dialog `New Class`.



Noua clasă `CInitials` ar trebui să figureze acum în secțiunea **ClassView**. Mai întâi va trebui să adăugați directiva `#include` menționată de `ClassWizard`. Pentru aceasta efectuați un dublu clic pe numele clasei `CEdit` în cadrul secțiunii **ClassView**. Fișierul antet al clasei `CEdit` va fi deschis într-o fereastră de editare. Modificați acest fișier antet adăugând o singură linie înaintea liniei care conține declarația clasei. Fișierul modificat ar trebui să arate astfel:

```
#include "Initials.h"

class CEditsDlg : public CDialog
{
...
}
```

Atunci când `ClassWizard` a creat clasa specificată, acesta a generat două noi fișiere și le-a adăugat în proiect: `Initials.h` și `Initials.cpp`. Directiva `#include` este necesară în `EditsDlg.h` deoarece clasa `CEdit` conține acum o variabilă membru de tipul `CInitials`, tip care este declarat în `Initials.h`. Omiterea directivei `#include` ar avea ca efect erori la compilare.

Acum puteți utiliza clasa `CInitials` pentru a implementa operația de introducere a inițialelor unei persoane. Prima sarcină este să respingeți caracterele care nu sunt litere, să transformați minusculele în majuscule și să adăugați automat un punct după fiecare literă.

Toate acestea pot fi realizate prin crearea unei funcții de tratare a mesajului WM_CHAR. Mesajul WM_CHAR este trimis de fiecare dată când este apăsat un caracter normal de pe tastatură în timp ce controlul de editare deține focusul. Spre deosebire de cazul mesajului EN_CHANGE, funcția de tratare pentru mesajul WM_CHAR primește caracterele înainte ca acestea să fie afișate, fiind astfel locul ideal pentru o operație de validare. În cazul în care caracterul introdus este corect, acesta poate fi transmis controlului de editare. Caracterele necorespunzătoare pot fi pur și simplu ignorate.

Crearea unei funcții subclasate de tratare a mesajelor prin intermediul ClassWizard

1. Apăsați Ctrl+W pentru a deschide ClassWizard sau selectați **ClassWizard** din meniul View.
2. Selectați pagina **Message Maps**.
3. Selectați **CInitials** în caseta combinată **Class Name**.
4. Selectați **WM_CHAR** în cadrul casetei cu listă **Messages**.
5. Efectuați un clic pe butonul **Add Function**. OnChar va apărea în lista **Member Functions** aflată în partea inferioară a casetei de dialog ClassWizard.
6. Efectuați un clic pe butonul **Edit Code**.

Vă aflați acum în cadrul funcției membru OnChar() a clasei CInitials. Această funcție va fi apelată la introducerea unui caracter în caseta de editare **Initials**. Parametrul nChar conține valoarea de tip întreg a caracterului introdus. Ceilalți parametri, nRepCnt și nFlags, nu prezintă importanță pentru exemplul nostru și nu trebuie să-i luați în seamă. Adăugați codul prezentat în listingul 5.5.

LISTINGUL 5.5 LST05_5.CPP – Funcția de tratare a mesajului OnChar

```
1 void CInitials::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
2 {
3     // TODO: Add your message handler code here and/or call default
4     // ** Verificam daca a fost introdusa o litera
5     if( isalpha(nChar) ) — ①
6     {
7         // ** Transformam minusculele in majuscule
8         if( islower(nChar) ) — ②
9             nChar -= 32;
10
11         // ** Apelam procedura de fereastră implicita
12         // ** deoarece s-ar putea ca valoarea nChar sa fie modificata
13         DefWindowProc(WM_CHAR, nChar, — ③
14             MAKELONG(nRepCnt, nFlags));
15
16         // ** Apelam din nou procedura de fereastră
17         // ** implicita pentru a adauga punctul
18         nChar = '.';
```

```
19         DefWindowProc(WM_CHAR, nChar,
20             MAKELONG(nRepCnt, nFlags));
21     }
22     // ** Daca a fost apasata tasta Backspace apelam
23     // ** de doua ori functia din clasa de baza pentru a sterge
24     // ** atat punctul cat si litera
25     if( nChar == VK_BACK ) — ④
26     {
27         CEdit::OnChar(nChar, nRepCnt, nFlags);
28         CEdit::OnChar(nChar, nRepCnt, nFlags);
29     }
30 }
```

- ① Verificăm dacă a fost introdusă o literă.
- ② Transformăm literele minuscule în majuscule.
- ③ Apelăm procedura de fereastră implicită pentru a asigura prelucrarea obișnuită a caracterului introdus. Caracterul devine apoi un punct și procedura de fereastră implicită este apelată din nou.
- ④ O apăsare a tastei Backspace va deveni o serie de două apăsări ale acestei taste.

Instrucțiunea if din linia 5 verifică dacă acel caracter trimis funcției prin intermediul parametrului nChar este o literă a alfabetului. În linia 8 se verifică dacă litera respectivă este minusculă. În caz că da, litera este transformată în majusculă scăzându-se 32 din codul său (diferența dintre literele minuscule și cele majuscule în cadrul setului de caractere ANSI este 32).

În linia 13 este apelată funcția DefWindowProc() (procedura de fereastră implicită). Motivul pentru care trebuie apelată această funcție, și nu funcția CEdit::OnChar() din clasa de bază, este faptul că aceasta din urmă nu percepe modificările aduse parametrului nChar. Dacă citiți informațiile referitoare la funcția CWnd::OnChar() veți vedea o observație care arată că funcția utilizează parametrii originali din mesaj, ignorând orice parametri transmiși explicit.

În linia 18 se înlocuiește valoarea curentă nChar cu codul unui punct, acesta fiind trimis către controlul de editare printr-un nou apel al funcției DefWindowProc().

În linia 25 se verifică dacă a fost apăsată tasta Backspace. VK_BACK reprezintă un nume simbolic care definește valoarea codului de tastă virtual pentru tasta Backspace. Astfel de definiții de coduri de taste virtuale există pentru toate tastele. Tabelul 5.3 înfățișează câteva exemple. În cazul în care a fost apăsată tasta Backspace, se apelează de două ori funcția CEdit::OnChar() a clasei de bază pentru a șterge atât punctul cât și litera.

Asamblați și lansați în execuție. Testați funcționalitatea clasei CInitials pe care ați creat-o.

TABELUL 5.3 Coduri de taste virtuale

Definiție	Tasta corespunzătoare
VK_BACK	Backspace
VK_TAB	Tasta Tab
VK_LEFT	Săgeata stânga
VK_RIGHT	Săgeata dreapta
VK_RETURN	Tasta Enter
VK_ESCAPE	Tasta Esc
VK_CONTROL	Tasta Ctrl

Utilizarea controalelor de editare multi-linie

Casetele de editare nu trebuie să conțină neapărat o singură linie. Pentru a crea o casetă de editare multi-linie nu trebuie decât să măriți înălțimea acesteia în cadrul machetei casetei de dialog și să selectați stilul **Multiline**. De asemenea, veți găsi probabil necesară selectarea stilului **Want Return**. Având acest stil selectat, puteți apăsa Enter în cadrul casetei de editare pentru a insera o nouă linie.

Limitele casetelor de editare multi-linie

Casetele de editare multi-linie sunt limitate la maximum 64 KB, adică aproximativ 64000 de caractere.

Casetele de editare multi-linie răspund la derularea verticală și pot crea automat bare de derulare atunci când liniile de text nu mai încap în cadrul controlului.

Din fericire, principiile de utilizare a casetelor de editare multi-linie sunt în general aceleași ca în cazul casetelor cu o singură linie. Veți folosi clasa `CEdit`, aceasta conținând mai multe funcții membru dedicate operării cu mai multe linii, funcții prezentate în tabelul 5.4.

TABELUL 5.4 Funcții membru ale clasei `CEdit` privind operarea cu mai multe linii

Numele funcției	Descriere
<code>GetLineCount()</code>	Furnizează numărul de linii.
<code>GetLine()</code>	Furnizează o linie de text.
<code>FmtLines()</code>	Activează sau dezactivează întreruperile de linie soft.
<code>SetTabStops()</code>	Permite specificarea pozițiilor de tabulare.

Ultimul tip de control de editare pe care vi-l menționez este controlul de editare formatată. Acest control este mult mai sofisticat decât o casetă de editare și permite formatare atât la nivel de caracter, cât și la nivel de paragraf. Controalele de editare formatată apar foarte rar în casetele de dialog; ele sunt de regulă folosite în cadrul reprezentărilor implementate de aplicații. Din această cauză nu vom discuta acum despre aceste controale.

VEDEȚI...

► Pentru detalii legate de controlul de editare formatată, vedeți pagina 449.

CAPITOLUL

6

Utilizarea controalelor de tip listă

Afișarea de liste cu informații în casetele de dialog

Păstrarea datelor într-o structură arborescentă

Operarea cu selecții singulare și multiple de elemente

Implementarea relațiilor de dependență între listele dintr-o casetă de dialog

Modul de prezentare a informațiilor în cadrul unui program are o influență determinantă asupra măsurii în care aplicația este prietenoasă cu utilizatorul. Cu cât o aplicație se dovedește mai confortabilă pentru utilizator, cu atât este mai probabil ca aceasta să aibă succes. O bună parte din informații vine sub formă de liste: directoare și fișiere, nume și stiluri de fonturi, poate chiar lista avioanelor care sunt pe punctul de a ateriza pe Otopeni.

Modalitatea optimă de afișare a unei liste de date depinde de semnificația informațiilor și de modul în care pot fi selectate acestea, individual sau în grup. Din acest motiv sunt disponibile mai multe tipuri de controale listă, atât textuale cât și grafice. Fiecare control are propriile caracteristici și mai multe stiluri care pot fi utilizate pentru a acoperi orice situație. Spre exemplu, o casetă combinată prezintă o listă sub o formă similară cu un meniu derulant, permițând selectarea unui singur element, în timp ce un control arbore poate fi folosit pentru a înfățișa o ierarhie de elemente având asociate imagini în fiecare nod.

Crearea controalelor de tip listă

Controalele de tip listă sunt oferite sub patru forme: casete combinate, casete cu listă, arbori și controale listă. Fiecare tip de control este destinat unui anumit scop în programare. La adăugarea controalelor listă în cadrul casetelor de dialog este importantă selectarea proprietăților de stil adecvate, deoarece acestea pot altera substanțial atât aspectul cât și comportamentul controlului. De pildă, selecția multiplă poate fi permisă atât pentru casetele cu listă cât și pentru controalele listă, iar stilul Drop List pentru casetele combinate activează funcționalitatea de casetă de editare, în timp ce stilul Dropdown o dezactivează.

Pe parcursul acestui capitol vom vedea cum se utilizează diferitele variante de controale de tip listă; dar mai întâi va trebui să apelați la AppWizard pentru a crea un nou proiect de tip dialog, numit Lists. Acest proiect va folosi toate cele patru feluri de controale de tip listă pentru a afișa informații despre directoare și fișiere.

VEDEȚI...

➤ Pentru detalii privind modul de creare a unei aplicații de tip dialog, vedeți pagina 54.

Adăugarea casetelor combinate

Un control casetă combinată se numește astfel deoarece reprezintă de fapt o combinație de controale: o casetă de editare, o casetă cu listă și un buton. Casetele combinate se folosesc pentru a afișa o listă de opțiuni, permițând selectarea uneia singure. Casetele combinate se disting între controalele de tip listă prin aceea că elementul selectat este întotdeauna vizibil, fiind afișat în partea superioară a controlului.

Există trei tipuri de casete combinate, așa cum o arată tabelul 6.1. Tipul unei casete combinate se stabilește în cadrul paginii **Styles** a casetei de dialog Combo Box Properties, la adăugarea controlului în macheta unei casete de dialog.

În exemplul nostru vom utiliza o casetă combinată pentru a permite selectarea unui director principal, care va fi apoi inspectat pentru a înscrice în celelalte controale din caseta de dialog informații privind subdirectoarele și fișierele conținute.

TABELUL 6.1 Tipuri de casete combinate

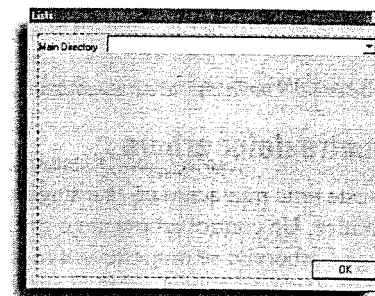
Tip de casetă	Descriere
Simplă	Îmbină o casetă de editare și o casetă cu listă. Lista este întotdeauna vizibilă și elementul selectat este afișat în caseta de editare.
Derulantă	Îmbină o casetă de editare cu un buton și o casetă cu listă. Casetă cu listă este vizibilă numai la efectuarea unui clic pe buton.
Listă derulantă	Îmbină o etichetă statică cu un buton și o casetă cu listă. Acest tip este similar cu caseta derulantă, deosebirea constând în faptul că utilizatorul nu poate introduce date în cadrul controlului.

Adăugarea unei casete combinate în caseta de dialog Lists

1. Deschideți caseta de dialog `IDD_LISTS_DIALOG` în cadrul editorului de resurse și înlăturați controlul etichetă `TODO`.
2. Înlăturați butonul **Cancel** și modificați poziția butonului **OK**, plasându-l în colțul dreapta jos al casetei de dialog. Va trebui să adăugați mai multe controale în această casetă de dialog, așadar sporiiți lățimea și înălțimea sa în concordanță cu figura 6.1.
3. Selectați pictograma etichetei statice de pe bara cu instrumente Controls, după care adăugați un control etichetă în partea superioară stânga a casetei de dialog.
4. Introduceți `Main Directory` pe post de etichetă.
5. Selectați pictograma casetei combinate de pe bara cu instrumente Controls, după care adăugați un astfel de control în partea dreaptă a etichetei **Main Directory**.
6. Extindeți lungimea casetei combinate până în marginea din dreapta a casetei de dialog.

FIGURA 6.1

Adăugarea unei casete combinate în caseta de dialog Lists.



7. Introduceți `IDC_MAIN_DIR` în caseta combinată **ID**.
8. Selectați pagina **Styles** și alegeți **Drop List** în cadrul casetei combinate **Type**. Casetă de dialog ar trebui să arate acum similară cu figura 6.1.

La adăugarea unei casete combinate într-o casetă de dialog, opțiunea **Sort** este selectată implicit. Aceasta înseamnă că elementele adăugate în caseta combinată vor fi afișate automat în ordine alfabetică. Pentru a inhiba acest comportament, selectați pagina **Styles** și deselectați opțiunea **Sort**. O altă necesitate uzuală în ceea ce privește casetele combinate

este modificarea dimensiunii casetei cu listă derulantă. Aceasta se realizează efectuând un clic pe săgeata din partea dreaptă a controlului. Astfel va fi afișat un cadru de formatare care indică dimensiunea casetei cu listă, acest cadru putând fi redimensionat prin intermediul butoanelor sale de dimensionare.

Odată adăugat controlul casetă combinată în cadrul casetei de dialog, apăsați la ClassWizard pentru a atașa o variabilă, așa cum o arată pașii următori:

Atașarea unei variabile de tip *CComboBox* la un control casetă combinată prin intermediul ClassWizard

1. Apăsați Ctrl+W pentru a apela ClassWizard sau selectați ClassWizard din meniul View.
2. Selectați pagina Member Variables.
3. Selectați CListsDlg din caseta combinată Class Name.
4. Selectați IDC_MAIN_DIR din caseta cu listă Control IDs.
5. Efectuați un clic pe butonul Add Variable; va fi afișată caseta de dialog Add Member Variable.
6. Asigurați-vă că în caseta combinată Category este selectat Control și că în Variable Type este selectat CComboBox.

Noua casetă combinată extinsă

Visual C++ 6 oferă o casetă combinată extinsă care vă permite adăugarea de imagini într-o casetă combinată normală.

7. Introduceți m_cbMainDir în caseta Member Variable Name și apoi efectuați clic pe OK.
8. Efectuați un clic pe OK pentru a închide ClassWizard.

VEDEȚI...

► Pentru amănunte privind noua casetă combinată extinsă, vedeți pagina 240.

Adăugarea controalelor arbore

Controlul arbore este unic prin aceea că este singurul control orientat anume înspre afișarea de informații ierarhice. Un control arbore are o structură stânga-dreapta. Un element din extremitatea stângă a arborelui se numește *nod rădăcină*. Un element din extremitatea dreaptă (care nu are elemente subordonate) se numește *nod frunză*, iar un element aflat între o rădăcină și o frunză se numește *nod ramificație*. Afișarea de linii care să conecteze elementele poate fi stabilită prin intermediul stilurilor. În mod implicit, controalele arbore permit selectarea unui singur element la un moment dat; dacă doriți să acordați utilizatorului posibilitatea de a selecta simultan mai multe elemente dintr-un arbore, veți fi nevoit să scrieți cod în acest sens.

În cadrul proiectului Lists va fi utilizat un control arbore pentru a afișa fișierele dintr-un director în ordine alfabetică. Veți crea câte un nod rădăcină pentru fiecare literă din alfabet și apoi veți insera elemente corespunzătoare fișierelor sub nodul adecvat.

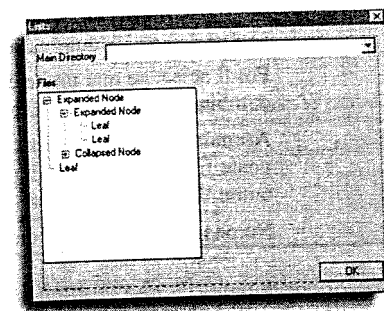
Pentru adăugarea unui control arbore în proiectul Lists urmați pașii de mai jos, iar apoi atașați la acest control o variabilă prin intermediul pașilor titrați „Atașarea unei variabile de tip CTreeCtrl la un control arbore prin intermediul ClassWizard”.

Adăugarea unui control arbore în caseta de dialog Lists

1. Deschideți caseta de dialog IDD_LISTS_DIALOG în cadrul editorului de resurse.
2. Adăugați un control etichetă statică, numit Files, pe post de titlu pentru controlul arbore, așa cum se vede în figura 6.2.
3. Selectați pictograma controlului arbore de pe bara cu instrumente Controls, după care adăugați un control arbore în partea stângă a casetei de dialog.
4. Introduceți IDC_FILES_TREE în caseta combinată ID.
5. Selectați pagina Styles și validați opțiunile Has Buttons, Has Lines și Lines at Root.
Caseta de dialog ar trebui să arate acum ca în figura 6.2.

FIGURA 6.2

Adăugarea unui control arbore în caseta de dialog Lists.



Atașarea unei variabile de tip *CTreeCtrl* la un control arbore prin intermediul ClassWizard

1. Apăsați Ctrl+W pentru a apela ClassWizard sau selectați ClassWizard din meniul View.
2. Selectați pagina Member Variables.
3. Selectați CListsDlg din caseta combinată Class Name.
4. Selectați IDC_FILES_TREE din caseta cu listă Control IDs.
5. Efectuați un clic pe butonul Add Variable; va fi afișată caseta de dialog Add Member Variable.
6. Asigurați-vă că în caseta combinată Category este selectat Control și că în Variable Type este selectat CTreeCtrl.
7. Introduceți m_treeFiles în caseta Member Variable Name și apoi efectuați clic pe OK.
8. Efectuați un clic pe OK pentru a închide ClassWizard.

VEDEȚI...

► Pentru mai multe informații despre stilurile controalelor arbore și despre utilizarea acestor controale în cadrul unei reprezentări, vedeți pagina 441.

Adăugarea controalelor casetă cu listă

În cea mai elementară formă a sa, o casetă cu listă constă într-o simplă listă de elemente, dar, spre deosebire de caseta combinată și de controlul arbore, o casetă cu listă poate permite atât selectarea elementelor individuale cât și selecții multiple. De fapt, sunt disponibile patru tipuri de selecție, prezentate în tabelul 6.2.

Controalele arbore și selectarea mai multor elemente

Clasa `CTreeCtrl` nu permite selectarea mai multor elemente simultan, dar este posibilă subclasarea acesteia în scopul adăugării funcționalității dorite.

TABELUL 6.2 Tipurile de selecție în cadrul casetelor cu listă

Tip de selecție	Descriere
Singulară	Nu poate fi selectat decât un singur element. Selectarea unui element duce la anularea selecției anterioare.
Multiplă	Pot fi selectate mai multe elemente utilizând mouse-ul în combinație cu tastele Shift și Ctrl.
Extinsă	Asemănătoare cu selecția multiplă, dar o mulțime de elemente poate fi selectată și prin încadrarea lor cu mouse-ul în timp ce butonul stâng este apăsat.
Dezactivată	Nu poate fi selectat nici un element.

Ca și în cazul casetelor combinate, în mod implicit este selectată ordonarea automată a elementelor. Pentru a dezactiva acest comportament, accesați pagina **Styles** și deselectați opțiunea **Sort**. Proiectul Lists folosește o casetă cu listă pentru a afișa o listă de subdirectoare. Dintre acestea pot fi selectate mai multe subdirectoare simultan, informații suplimentare despre elementele selectate fiind afișate într-un control listă. Pentru a adăuga în proiectul Lists o casetă cu listă cu selecție multiplă, urmați pașii aflați sub titlul „Adăugarea unui control casetă cu listă în caseta de dialog Lists”. Aici se face referire la opțiunea de stil **No Integral Height**, care este deselectată pentru ca înălțimea controlului să fie astfel calculată încât să fie afișat un număr întreg de elemente. Dacă lăsați această opțiune selectată, este posibil ca în control să fie afișate elemente parțial.

Adăugarea unui control casetă cu listă în caseta de dialog Lists

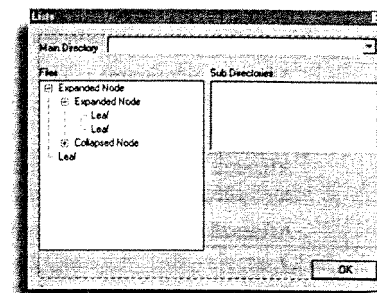
1. Deschideți caseta de dialog `IDD_LISTS_DIALOG` în cadrul editorului de resurse.
2. Adăugați un control etichetă statică, numit Sub Directories, cu rolul de titlu pentru controlul casetă cu listă (vedeți figura 6.3).
3. Selectați pictograma casetei cu listă de pe bara cu instrumente Controls, după care adăugați o casetă cu listă pe suprafața casetei de dialog, așa cum se vede în figura 6.3.
4. Introduceți `IDC_SUB_DIRS` în caseta combinată **ID**.
5. Accesați pagina **Styles** și selectați **Extended** din caseta combinată **Selection**.

6. În cadrul paginii **Styles**, deselectați opțiunea **No Integral Height**. Caseta de dialog ar trebui să arate acum ca în figura 6.3.

Odată adăugată caseta cu listă, atașați la aceasta o variabilă.

FIGURA 6.3

Adăugarea unei casete cu listă în cadrul casetei de dialog Lists.



Atașarea unei variabile de tip `CListBox` la un control casetă cu listă prin intermediul ClassWizard

1. Apăsați Ctrl+W pentru a apela ClassWizard sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Member Variables**.
3. Selectați `CLISTS_DIALOG` din caseta combinată **Class Name**.
4. Selectați `IDC_SUB_DIRS` din caseta cu listă **Control IDs**.
5. Efectuați un clic pe butonul **Add Variable**; va fi afișată caseta de dialog **Add Member Variable**.
6. Asigurați-vă că în caseta combinată **Category** este selectat **Control** și că în **Variable Type** este selectat `CListBox`.
7. Introduceți `m_lbSubDirs` în caseta **Member Variable Name** și apoi efectuați clic pe **OK**.
8. Efectuați un clic pe **OK** pentru a închide ClassWizard.

Adăugarea unui control listă

Controlul listă are o denumire ușor derutantă, pentru că celelalte trei controale prezentate în acest capitol sunt de asemenea controale de tip listă. Și totuși, controlul listă este cel mai complex dintre cele patru și este utilizat mai degrabă în cadrul unei reprezentări decât în casetele de dialog. Un control listă este în stare să afișeze atât pictograme, cât și etichete asociate, și poate opera în oricare din cele patru moduri descrise în tabelul 6.3.

Windows Explorer utilizează un control listă

Exemplul cel mai celebru de control listă se găsește în Windows Explorer. Meniul **View** ilustrează cele patru moduri de afișare diferite.

TABELUL 6.3 Modurile de reprezentare ale controlului listă

Mod	Descriere
Pictograme	Afișează pictograme mari (32x32 pixeli) cu etichete plasate sub fiecare pictogramă. Elementele sunt plasate în ordine de la stânga la dreapta și apoi de sus în jos.
Pictograme mici	Afișează pictograme mici (16x16 pixeli) cu etichete plasate în dreapta fiecărei pictograme. Elementele sunt plasate în ordine de la stânga la dreapta și apoi de sus în jos.
Listă	Afișarea este similară cu modul cu pictograme mici, dar elementele sunt plasate în ordine de sus în jos și apoi de la stânga la dreapta.
Raport	Afișează informații prin intermediul unor coloane cu antet.

Menținerea vizibilă a selecției

În mod implicit, atunci când un control listă pierde focusul, elementul selectat nu mai este afișat evidențiat. Dacă doriți ca selecția să rămână vizibilă, validați stilul **Show Selection Always**.

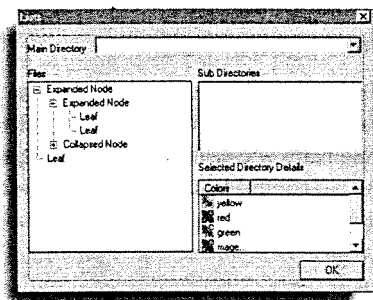
În cadrul proiectului Lists, controlul listă va fi utilizat pentru a afișa detalii privind directoarele selectate în caseta cu listă aflată deasupra sa în caseta de dialog. Controlul listă va avea trei coloane; prima coloană va conține numele directorului, cea de a doua va afișa numărul de fișiere conținute, iar a treia coloană va înfățișa dimensiunea în megaocteti a directorului. Pentru adăugarea controlului listă urmați pașii de mai jos. După aceea atașați o variabilă la acest control, parcurgând pașii titrați „Atașarea unei variabile de tip `CListCtrl` la o casetă cu listă prin intermediul ClassWizard”.

Adăugarea unui control listă în caseta de dialog Lists

1. Deschideți caseta de dialog `IDD_LISTS_DIALOG` în cadrul editorului de resurse.
2. Adăugați un control etichetă statică, numit Selected Directories Details, ca și titlu pentru controlul listă (vedeți figura 6.4).

FIGURA 6.4

Adăugarea unui control listă în cadrul casetei de dialog Lists.



3. Selectați pictograma controlului listă de pe bara cu instrumente Controls, după care adăugați acest control pe suprafața casetei de dialog, așa cum se vede în figura 6.4.

4. Introduceți `IDC_SELECTED_DIRS` în caseta combinată **ID**.

5. Accesați pagina **Styles** și selectați **Report** în cadrul casetei combinate **View**. Caseta de dialog ar trebui să arate acum ca în figura 6.4.

Atașarea unei variabile de tip `CListCtrl` la un control listă prin intermediul ClassWizard

1. Apăsăți **Ctrl+W** pentru a apela ClassWizard sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Member Variables**.
3. Selectați `CListsDlg` din caseta combinată **Class Name**.
4. Selectați `IDC_SELECTED_DIRS` din caseta cu listă **Control IDs**.
5. Efectuați un clic pe butonul **Add Variable**; va fi afișată caseta de dialog **Add Member Variable**.
6. Asigurați-vă că în caseta combinată **Category** este selectat **Control** și că în **Variable Type** este selectat `CListCtrl`.
7. Introduceți `m_lcDirDetails` în caseta **Member Variable Name** și apoi efectuați clic pe **OK**.
8. Efectuați un clic pe **OK** pentru a închide ClassWizard.

VEDEȚI...

- Pentru mai multe informații despre stilurile controalelor listă și despre utilizarea acestor controale în cadrul unei reprezentări, vedeți pagina 427.

Adăugarea de elemente în controalele de tip listă

Deoarece controalele listă sunt orientate înspre afișarea și selecția elementelor de informație, prima cerință în programare este popularea acestor controale cu date. Fiecare informație adăugată într-un control de tip listă reprezintă un element. Deși mecanismele de inserare a elementelor prezintă similarități, veți vedea că fiecare clasă de control are anumite particularități.

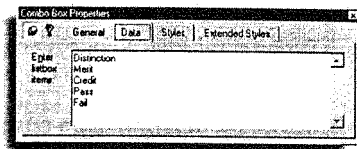
Popularea unei casete combinate

Caseta combinată este singurul dintre cele patru controale de tip listă care poate fi populat cu elemente în cadrul editorului de resurse. Aceasta se realizează prin intermediul paginii **Data** a casetei de dialog **Combo Box Properties**. Un exemplu este înfățișat în figura 6.5. Astfel, fiecare element poate fi introdus în caseta **Enter Listbox Items**. Dacă efectuați la un moment dat această operație, nu uitați să apăsați **Ctrl+Enter** după fiecare element, pentru că simpla apăsare a tastei **Enter** va duce la închiderea casetei de dialog. O astfel de populare a unei casete combinate nu este uzuală; în mod normal, casetele combinate sunt populate la momentul execuției, de multe ori în cadrul funcției `OnInitDialog()`. Infrastructura MFC apelează această funcție la deschiderea casetei de dialog, înainte de afișarea ei.

Clasa MFC care încapsulează casetele combinate este `CComboBox`. Această clasă conține mai multe funcții care se ocupă de adăugarea și înlăturarea elementelor; acestea sunt prezentate în tabelul 6.4. Fiecărui element îi este asociat la adăugare un indice numeric pornind de la zero, acest indice putând fi utilizat apoi pentru accesarea respectivului element.

FIGURA 6.5

Pagina Data a casetei de dialog Combo Box Properties.



TABELUL 6.4 Funcții de populare din clasa CComboBox

Numele funcției	Descriere
AddString	Adaugă un element fie la sfârșitul listei, fie în poziția corespunzătoare ordonării.
DeleteString	Înlătură un element.
InsertString	Inserează un element într-o anumită poziție.
ResetContent	Înlătură toate elementele existente.
Dir	Metodă specială de populare pentru inserarea ca elemente a numelor de fișiere.

În cazul proiectului Lists, caseta combinată este populată în cadrul funcției OnInitDialog() a casetei de dialog, fiind inserate o serie de directoare principale obținute prin intermediul unor funcții globale din Windows. În acest scop, creați mai întâi o funcție nouă având numele PopulateCombo() și tipul void. Apoi modificați OnInitDialog() pentru a efectua apelul noii funcții după comentariile //TODO:, așa cum o arată linia 28 a listingului 6.1. După aceea adăugați liniile de cod de la 34 până la 53, acestea efectuând inserarea de elemente în caseta combinată.

LISTINGUL 6.1 LST06_1.CPP – Popularea casetei combinate

```

1  BOOL CListsDlg::OnInitDialog() ← fct. apelată automat
2  {
3      CDialog::OnInitDialog();
4
5      // Add "About..." menu item to system menu.
6      // IDM_ABOUTBOX must be in the system command range.
7      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
8      ASSERT(IDM_ABOUTBOX < 0xF000);
9
10     CMenu* pSysMenu = GetSystemMenu(FALSE);
11     if (pSysMenu != NULL)
12     {
13         CString strAboutMenu;
14         strAboutMenu.LoadString(IDS_ABOUTBOX);
15         if (!strAboutMenu.IsEmpty())
16         {
17             pSysMenu->AppendMenu(MF_SEPARATOR);

```

```

18         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
19                               strAboutMenu);
20     }
21     // Set the icon for this dialog box.
22     // The framework does this automatically
23     // when the application's main window
24     // is not a dialog box
25     SetIcon(m_hIcon, TRUE); // Set big icon
26     SetIcon(m_hIcon, FALSE); // Set small icon
27
28     // TODO: Add extra initialization here
29     // ** Initializăm caseta combinată cu directoare principale
30     PopulateCombo();
31     return TRUE; // return TRUE unless you set the focus
32     // to a control
33 }
34 void CListsDlg::PopulateCombo()
35 {
36     TCHAR szBuffer[MAX_PATH];
37
38     // ** Aflăm directorul Windows, de regula C:\Windows,
39     // ** și îl adăugăm în caseta combinată
40     GetWindowsDirectory(szBuffer, MAX_PATH);
41     m_cbMainDir.AddString(szBuffer);
42
43     // ** Scurtăm directorul și lasăm doar litera de unitate C:
44     // ** pe care o adăugăm în caseta combinată
45     szBuffer[2]=0;
46     m_cbMainDir.AddString(szBuffer);
47
48     // ** Aflăm directorul System,
49     // ** de regula, C:\Windows\System
50     // ** și-l adăugăm în caseta combinată
51     GetSystemDirectory(szBuffer, MAX_PATH);
52     m_cbMainDir.AddString(szBuffer);
53
54     // ** Aflăm directorul curent și-l adăugăm în caseta combinată
55     GetCurrentDirectory(MAX_PATH, szBuffer);
56     m_cbMainDir.AddString(szBuffer);
57 }

```

① Apelăm funcția PopulateCombo în cadrul funcției OnInitDialog.

② Adăugăm directorul Windows în caseta combinată.

③ Reducem șirul de caractere la litera de unitate și îl adăugăm sub această formă în caseta combinată.

4 Adăugăm directorul System în caseta combinată.

5 Adăugăm directorul curent în caseta combinată.

Apelurile de funcții `GetWindowsDirectory()`, `GetSystemDirectory()` și `GetCurrentDirectory()` din liniile 38, 48 și, respectiv, 52 sunt apeluri de funcții globale care înscriu în bufferul șir de caractere `szBuffer` trimis ca parametru căile de directoare respective. Macrodefiniția `MAX_PATH` este folosită pentru a specifica lungimea maximă a unei căi; în majoritatea cazurilor, această lungime este 260. După obținerea fiecărei căi, `szBuffer` este transmis funcției `AddString()` a variabilei membru `m_cbMainDir`, aceasta ocupându-se de inserarea elementului în cadrul casetei combinate. Deoarece caseta combinată are stilul `Sort`, elementele inserate vor fi dispuse automat în ordine alfabetică.

Tratarea mesajelor corespunzătoare casetelor combinate

Scopul unei casete combinate este să permită selectarea unuia dintre elementele pe care le conține. Atunci când aceasta se întâmplă natural, programul trebuie să afle imediat și să răspundă la modificarea selecției. Acest lucru este posibil prin interceptarea mesajului `CBN_SELCHANGE`, trimis de către control casetei de dialog oricând se modifică elementul selectat.

Testați apariția erorilor la popularea unei casete combinate

Este bine să verificați valorile întoarse de funcțiile `AddString()` și `InsertString()`. În cazul apariției unei erori, aceste funcții întorc una din valorile `CB_ERR` sau `CB_ERRSPACE`.

Adăugați o funcție de tratare a mesajului `CBN_SELCHANGE` pentru caseta combinată `IDC_MAIN_DIR`. Astfel ar trebui să fie creată funcția membru `OnSelchangeMainDir()`. Veți avea nevoie, totodată, de o nouă variabilă membru de tip `CString`, `m_strMainDir`, care va păstra calea selectată. `m_strMainDir` va fi utilizată în secțiunile următoare pentru a popula controlul arbore și caseta cu listă. Acum adăugați codul din listingul 6.2.

LISTINGUL 6.2 LST06_2.CPP – Extragerea textului din elementul selectat în cadrul casetei combinate

```
1 void CListsDlg::OnSelchangeMainDir() — 1
2 {
3     // TODO: Add your control notification handler code here
4     // ** Extragem indicele elementului selectat
5     int nIndex = m_cbMainDir.GetCurSel(); — 2
6
7     // ** Verificam daca indicele este valid
8     if(nIndex != CB_ERR)
9     {
10         // ** Extragem textul din elementul selectat si-l plasam
11         // ** intr-o variabila membru, dupa care apelam functii
```

```
12         // ** care populeaza celelalte controale
13         m_cbMainDir.GetLBText(nIndex, m_strMainDir); — 3
14         PopulateTree(); — 4
15     }
16 }
```

1 Apelată la modificarea selecției în cadrul casetei combinate.

2 Extragem indicele directorului selectat.

3 Extragem și stocăm numele directorului selectat.

4 Apelăm o funcție care populează controlul arbore.

La primirea mesajului că elementul selectat s-a modificat, în linia 5 este apelată funcția `GetCurSel()` care întoarce în `nIndex` indicele noului element selectat. Dacă nu este selectat nici un element, este întoarsă valoarea specială `CB_ERR`, eventualitate testată în linia 8. Dacă `nIndex` este valid, este transmis în linia 13 funcției `GetLBText()` împreună cu variabila membru `m_strMainDir`, aceasta din urmă fiind modificată. Puteți observa că linia 14 conține apelul unei noi funcții, `PopulateTree()`; această funcție o vom crea în secțiunea care urmează.

Popularea unui arbore

La popularea unui arbore, de multe ori este necesară stocarea de informații privind structura arborelui pentru a putea insera elemente ca părinți sau ca subordonați ai altor elemente introduse anterior.

Clasa MFC care încapsulează un arbore este `CTreeCtrl`; această clasă include funcțiile prezentate în tabelul 6.5, funcții care se ocupă de inserarea și înlăturarea de elemente. Funcția `InsertItem()` are mai multe versiuni supradefinite pentru a putea să realizeze asocieri de imagini și de pointeri la date externe arborelui, dacă este nevoie. Fiecărui element inserat îi este asociat un indicator de tip `HTREEITEM` care poate fi transmis funcției `InsertItem`, făcând astfel posibilă crearea unei ierarhii.

TABELUL 6.5 Funcții de populare din clasa `CTreeCtrl`

Numele funcției	Descriere
<code>InsertItem</code>	Inserează un element, fie ca element rădăcină, fie ca element subordonat, în funcție de parametrii transmiși.
<code>DeleteItem</code>	Înlătură un element.
<code>DeleteAllItems</code>	Înlătură toate elementele existente.

Pentru a popula arborele din proiect, creați mai întâi o nouă funcție membru numită `PopulateTree()` care întoarce tipul `void`. Arborele va afișa fișierele aflate în directorul selectat în cadrul casetei combinate, iar acestea vor fi împărțite alfabetic. Funcția inserează mai întâi câte un element pentru fiecare literă a alfabetului, după care adaugă un al 27-lea element care va subordona fișierele ale căror nume nu încep cu o literă. Sunt folosite apoi

o serie de funcții oferite de Windows pentru a inspecta conținutul directorului și a insera numele fiecărui fișier în subordinea elementului corespunzător din arbore. După ce ați creat funcția `PopulateTree()`, adăugați codul prezentat în listingul 6.3.

LISTINGUL 6.3 LST06_3.CPP – Popularea unui control arbore

```
1 void CListsDlg::PopulateTree()
2 {
3     // ** Inlaturam toate elementele existente in arbore
4     m_treeFiles.DeleteAllItems();
5     // ** Alocam un vector de elemente HTREEITEMS
6     HTREEITEM hLetter[27];
7     // ** Inseram ca elemente radacina literele de la 'A' la 'Z'
8     for(int nChar = 'A'; nChar <= 'Z'; nChar++)
9         hLetter[nChar - 'A'] =
10             m_treeFiles.InsertItem((TCHAR*)&nChar);
11
12     // ** Inseram elementul 'Other' ca element radacina
13     hLetter[26] = m_treeFiles.InsertItem("Other");
14
15     HANDLE hFind;
16     WIN32_FIND_DATA dataFind;
17     BOOL bMoreFiles = TRUE;
18     CString strFile;
19
20     // ** Cautam primul fisier din directorul principal
21     hFind = FindFirstFile(m_strMainDir + "\\*.*", &dataFind);
22
23     // ** Repetam pana cand am gasit toate fisierele
24     while(hFind != INVALID_HANDLE_VALUE && bMoreFiles == TRUE)
25     {
26         // ** Ne asiguram ca am gasit un fisier, nu un director
27         if(dataFind.dwFileAttributes ==
28             FILE_ATTRIBUTE_ARCHIVE)
29         {
30             // ** Aflam prima litera din numele fisierului
31             int nChar = dataFind.cFileName[0];
32             // ** Transformam literele minuscule in majuscule
33             if(islower(nChar))
34                 nChar -= 32;
35             // ** Daca numele fisierului incepe cu o litera atunci
36             // ** scadem 'A' pentru a afla indicele in cadrul
37             // ** vectorului hLetter, altfel folosim indicele 26
38             if(isalpha(nChar))
39                 nChar -= 'A';
40             else
41                 nChar = 26;
```

```
40         // ** Inseram numele fisierului in arbore
41         m_treeFiles.InsertItem(dataFind.cFileName,
42             hLetter[nChar]);
43     }
44     // ** Cautam urmatorul fisier din directorul principal
45     bMoreFiles = FindNextFile(hFind, &dataFind);
46 }
47 // ** Inchidem indicatorul de cautare
48 FindClose(hFind);
49 }
```

- 1 Adăugăm elementele rădăcină (literele alfabetului).
- 2 Căutăm un fișier din directorul `m_strMainDir`.
- 3 Parcurgem fișierele din directorul principal.
- 4 Inserăm numele fișierului în arbore.

În linia 4, apelul funcției `DeleteAllItems()` înlătură toate elementele existente; această operație este necesară deoarece arborele este populat de fiecare dată când se modifică directorul selectat în cadrul casetei combinate. Bucla `for` din liniile 8 și 9 apelează funcția `InsertItem()`, transmitând pe rând literele de la A la Z; deoarece nu sunt transmiși alți parametri, literele sunt inserate automat în arbore ca elemente rădăcină. La fiecare apel al funcției `InsertItem()`, indicatorul `HTREEITEM` întors este stocat în vectorul `hLetter`; așadar, `hLetter[0]` conține indicatorul lui A, `hLetter[1]` conține indicatorul lui B și așa mai departe. În linia 12 este inserat un element rădăcină "Other" care va subordona fișierele ale căror nume nu încep cu o literă.

Pentru a obține o listă a fișierelor este apelată mai întâi funcția `FindFirstFile()`, în linia 20. Parametrul `m_strMainDir + "\\*.*"` transmis acestei funcții va duce la căutarea tuturor fișierelor aflate în directorul selectat. Căutarea următorului fișier se face prin apelul `FindNextFile()` din linia 44. Această funcție întoarce `FALSE` atunci când au fost epuizate toate fișierele, ceea ce va duce la încheierea buclei. Ambele funcții `Find` înscriu datele dorite în variabila `dataFind`, care este o structură `WIN32_DATA_FIND`. Această structură conține informații despre fiecare fișier găsit.

În linia 26 este testat atributul de tip al fișierului găsit pentru a se asigura că este vorba despre un fișier obișnuit, nu despre un director. Prima literă a numelui de fișier este aflată în linia 29 și transformată eventual în majusculă în linia 32. Testul din linia 36 verifică dacă numele fișierului începe cu o literă, caz în care traduce valoarea întreagă a literei într-un indice în vectorul `hLetter`. În apelul `InsertItem()` din linia 41 sunt transmise numele fișierului care trebuie inserat și indicatorul `HTREEITEM` al părintelui care-l va subordona.

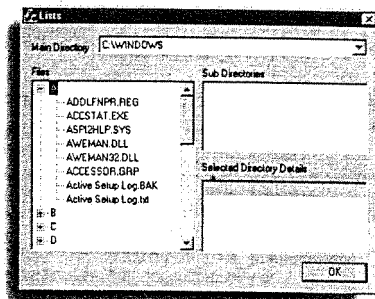
Asocierea unui element din arbore cu date interne ale aplicației

Uneori este la îndemână să asociați fiecare element din arbore cu un pointer la un obiect. Acest lucru se poate face prin intermediul funcțiilor `SetItemData()` și `GetItemData()`.

Asamblați și rulați aplicația. Caseta combinată ar trebui să afișeze mai multe directoare, iar selectarea unuia dintre acestea ar trebui să ducă la popularea arborelui cu fișierele conținute, așa cum se vede în figura 6.6.

FIGURA 6.6

Proiectul Lists având o casetă combinată și un arbore funcțional.



Popularea unei casete cu listă

Popularea unei casete cu listă se realizează practic la fel cu popularea unei casete combinate, deoarece elementele din lista unei casete combinate sunt de fapt elemente ale unei casete cu listă. Diferența esențială între cele două controale este modul de afișare și faptul că în casetele cu listă este posibilă selectarea mai multor elemente simultan.

Testați apariția erorilor la popularea unei casete cu listă

Este bine să verificați valorile întoarse de funcțiile `AddString()` și `InsertString()`. În cazul apariției unei erori, aceste funcții întorc una din valorile `CB_ERR` sau `CB_ERRSPACE`.

Clasa MFC care încapsulează casetele cu listă este `CListBox`. Funcțiile care răspund de inserarea și înlăturarea elementelor sunt identice cu cele prezentate în tabelul 6.4, mai devreme în acest capitol.

În cadrul proiectului Lists, caseta cu listă este populată cu numele subdirectoarelor aflate în directorul principal care a fost selectat în caseta combinată. În acest scop veți adăuga mai întâi o nouă funcție membru care se numește `PopulateListBox()` și are tipul `void`. Modificați apoi funcția `OnSelchangeMainDir()` pentru a apela această nouă funcție după apelul `PopulateTree()`.

Acum adăugați codul înfățișat în listingul 6.4, care realizează popularea casetei cu listă cu numele subdirectoarelor.

LISTINGUL 6.4 LST06_4.CPP – Popularea unei casete cu listă

```
1 void CListsDlg::PopulateListBox()
2 {
3     // ** Inlaturam toate elementele existente in caseta cu lista
4     m_lbSubDirs.ResetContent(); ❶
5 }
```

```
6 HANDLE hFind;
7 WIN32_FIND_DATA dataFind;
8 BOOL bMoreFiles = TRUE;
9
10 // ** Cautam primul fisier din directorul principal
11 hFind = FindFirstFile(m_strMainDir + "\\*.*", &dataFind); ❷
12
13 // ** Repetam pana cand am gasit toate fisierele
14 while(hFind != INVALID_HANDLE_VALUE && ❸
15     bMoreFiles == TRUE)
16 {
17     // ** Verificam daca am gasit un director
18     if(dataFind.dwFileAttributes ==
19         FILE_ATTRIBUTE_DIRECTORY) ❹
20     {
21         // ** Adaugam numele directorului in caseta cu lista,
22         // ** ignorand directoarele "." si ".."
23         if(strcmp(dataFind.cFileName, ".")
24             if(strcmp(dataFind.cFileName, ".."))
25             m_lbSubDirs.AddString(dataFind.cFileName);
26     }
27     // ** Cautam urmatorul fisier din directorul principal
28     bMoreFiles = FindNextFile(hFind, &dataFind);
29 FindClose(hFind);
30 }
```

❶ Ștergem caseta cu listă.

❷ Căutăm un fișier din directorul `m_strMainDir`.

❸ Parcurgem fișierele din directorul principal.

❹ Inserăm numele directorului în caseta cu listă, ignorând directoarele "." și "..".

Apelul `ResetContent()` din linia 4 înlătură toate elementele existente; această operație este necesară deoarece arborele este populat de fiecare dată când se modifică directorul selectat în cadrul casetei combinate. Pentru amănunte privind funcțiile `FindFirstFile()` și `FindNextFile()`, reveniți la secțiunea „Popularea unui arbore” aflată mai devreme în acest capitol.

În linia 17 este testat atributul de tip al fișierului găsit pentru a se asigura că este vorba despre un director. Cele două apeluri ale funcției `strcmp()` din liniile 21 și 22 au ca efect ignorarea directoarelor speciale "." și "..". În apelul `AddString()` din linia 23 este transmis numele fișierului (în acest caz, numele subdirectorului) care trebuie inserat în caseta cu listă. Dacă asamblați și rulați aplicația acum, ar trebui să vedeți cum selectarea unui director în caseta combinată duce la popularea atât a arborelui cât și a casetei cu listă.

Tratarea mesajelor corespunzătoare casetelor cu listă

Acțiunile efectuate de utilizator asupra controalelor casetă cu listă au ca efect transmiterea unor mesaje corespunzătoare către caseta de dialog. Mesajul `LBN_SELCHANGE` este trimis în cazul în care elementul sau (în cazul controalelor care permit selecția multiplă) elementele selectate se modifică. Modul în care se operează cu o selecție diferă după cum controlul permite sau nu selecția multiplă, iar clasa `CListBox` conține funcții adaptate fiecărui caz.

CStringList

`CStringList` este un exemplu de clasă MFC ajutătoare. Ea este folosită pentru administrarea unei liste variabile de obiecte individuale `CString`.

Caseta cu listă din proiectul `Lists` permite selecția multiplă. Funcția de tratare a mesajului creează o listă a directoarelor selectate în cadrul unei noi variabile membru, `m_strList`, care este de tip `CStringList`. Pentru început, adăugați această nouă variabilă prin intermediul casetei de dialog `Add Member Variable`.

Adăugați o funcție pentru tratarea mesajului `LBN_SELCHANGE` corespunzător casetei cu listă `IDC_SUB_DIRS`. Ar trebui să fie creată funcția membru `OnSelchangeSubDirs()`. Această funcție va determina numărul elementelor selectate și apoi va extrage numele fiecărui element și-l va adăuga în variabila `m_strList`. În fine, funcția va apela o nouă funcție, `PopulateListControl()`, care va utiliza `m_strList` pentru a introduce informații în controlul listă. Acum adăugați codul ilustrat în listingul 6.5.

LISTINGUL 6.5 LST06_5.CPP – Tratarea mesajelor trimise de caseta cu listă

```
1 void CListsDlg::OnSelchangeSubDirs() — ①
2 {
3     // TODO: Add your control notification handler code here
4     // ** Determinam numarul elementelor selectate
5     int nSelCount = m_lbSubDirs.GetSelCount(); — ②
6
7     // ** Resetam lista de siruri
8     m_strList.RemoveAll();
9     if(nSelCount)
10    {
11        CString str;
12        // ** Cream un vector de intregi care va stoca indici
13        // ** si-l initializam cu indicii elementelor selectate
14        LPINT pItems = new int[nSelCount];
15        m_lbSubDirs.GetSelItems(nSelCount, pItems);
16
17        for(int i = 0; i < nSelCount; i++) — ③
18        {
19            // ** Extragem numele elementului selectat
20            // ** si-l depunem intr-o lista de siruri
```

```
21        m_lbSubDirs.GetText(pItems[i], str);
22        m_strList.AddTail(str); — ④
23    }
24    // ** Stergem vectorul de intregi
25    delete [] pItems;
26 }
27 // ** Acum populam controlul lista
28 PopulateListControl(); — ⑤
29 }
```

- ① Apelată la modificarea selecției din cadrul casetei cu listă.
- ② Extragem numărul de elemente selectate în cadrul casetei cu listă.
- ③ Completăm vectorul pe baza casetei cu listă.
- ④ Adăugăm numele subdirectorului selectat în variabila listă de șiruri `m_strList`.
- ⑤ Apelăm funcția care populează controlul listă.

Funcția `GetSelCount()` apelată în linia 5 întoarce numărul elementelor selectate și este folosită doar pentru controalele care permit selecția multiplă. În linia 8, toate șirurile stocate în variabila `m_strList` sunt șterse prin intermediul funcției `RemoveAll()` a clasei `CStringList`, lăsând variabila pregătită pentru regenerare.

În linia 14 este alocat spațiu pentru un vector de întregi; acest vector va conține indicii elementelor selectate și, prin urmare, dimensiunea sa trebuie să fie egală cu `nSelCount`. Variabila `pItems` este un pointer la tipul `int` și este inițializată ca pointer la începutul vectorului. Observați că spațiul ocupat de vector este eliberat în linia 25, prin intermediul `delete []`, deoarece nu mai este necesar.

În linia 15, funcția `GetSelItems()` extrage din cadrul controlului lista elementelor selectate. Elementele acestei liste se consideră indexate de la zero. Cel de-al doilea parametru este adresa vectorului de întregi în care funcția va înscrie datele solicitate.

Bucloa `for` din linia 17 parcurge elementele selectate. La fiecare iterație este apelată `GetText()`, fiindu-i transmise următorul indice din vector (`pItems[i]`) și variabila `str` în care va fi plasat textul. Acest text este adăugat apoi, în linia 22, la sfârșitul variabilei `m_strList`. Pentru că datele se vor schimba de fiecare dată când se modifică lista subdirectoarelor selectate, elementele afișate în controlul listă vor fi actualizate prin apelul `PopulateListControl()` din finalul funcției, în linia 28.

Popularea unui control listă

Modalitatea de populare a unui control listă diferă puțin de metodele aplicate în cazul celorlalte controale de tip listă despre care am discutat în acest capitol. De asemenea, apar diferențe în funcție de tipul de control listă care este utilizat; cu totul există patru tipuri (pictograme, pictograme mici, listă și raport), prezentate în secțiunea „Adăugarea unui control listă”, mai devreme în acest capitol. Probabil că tipul cel mai întâlnit (și util) este

tipul raport. Acest tip se caracterizează prin afișarea informațiilor pe coloane. Spre exemplu, reprezentarea de tip raport din Explorer afișează coloanele titrate Name, Size, Type și Modified, informațiile respective despre fișiere fiind afișate linie după linie.

Clasa MFC care încapsulează controalele listă este `CListCtrl`; această clasă include funcțiile prezentate în tabelul 6.6, funcții care se ocupă de adăugarea și înlăturarea elementelor.

TABELUL 6.6 Funcții de populare din clasa `CListCtrl`

Numele funcției	Descriere
<code>InsertColumn</code>	Adaugă o nouă coloană într-o anumită poziție
<code>DeleteColumn</code>	Înlătură o coloană
<code>InsertItem</code>	Adaugă un nou element
<code>DeleteItem</code>	Înlătură un element existent
<code>DeleteAllItems</code>	Înlătură toate elementele existente
<code>SetItemText</code>	Stabilește conținutul textual al unei părți dintr-un element

Proiectul Lists folosește un control listă de tip raport care conține coloane pentru numele directorului, numărul de fișiere aflate în director și dimensiunea totală a acestor fișiere. Primul lucru care trebuie făcut este inserarea coloanelor; deoarece ele rămân neschimbate, această operație poate fi efectuată în cadrul funcției `OnInitDialog()`. Elementele propriu-zise vor fi adăugate în control de către funcția `PopulateListControl()`, așa cum am precizat în secțiunea precedentă. Acum este momentul să adăugați această funcție, specificând void ca tip întors.

În continuare adăugați în funcția `OnInitDialog()` liniile de cod de la 8 la 11 din listingul 6.6.

LISTINGUL 6.6 LST06_6.CPP – Inițializarea coloanelor din cadrul controlului listă

```

1 BOOL CListsDlg::OnInitDialog()
2 {
3
4 ...
5     // TODO: Add extra initialization here
6     // ** Initializam caseta combinata cu directoare principale
7     PopulateCombo();
8     // ** Initializam coloanele controlului lista
9     m_lcDirDetails.InsertColumn(0, "Directory", LVCFMT_LEFT, 70);
10    m_lcDirDetails.InsertColumn(1, "Files", LVCFMT_RIGHT, 50);
11    m_lcDirDetails.InsertColumn(2, "Size KB", LVCFMT_RIGHT, 60);
12
13    return TRUE; // return TRUE unless you set the focus to a
14    ↪control
15 }
```

❶ Inițializăm controlul listă cu trei coloane.

Primul parametru din apelul `InsertColumn()` reprezintă indicele numeric al coloanei, începând de la zero. Puteți opta între modurile de aliniere a textului `LVCFMT_LEFT`, `LVCFMT_RIGHT` și `LVCFMT_CENTER`, iar ultimul parametru specifică lățimea inițială a coloanei, exprimată în pixeli. Lățimea poate fi ajustată cu ajutorul mouse-ului în orice moment. Acum adăugați codul prezentat în listingul 6.7 pentru a efectua popularea completă a controlului.

LISTINGUL 6.7 LST06_7.CPP – Popularea unui control listă

```

1 void CListsDlg::PopulateControl()
2 {
3     // ** Înlăturăm toate elementele existente în controlul lista
4     m_lcDirDetails.DeleteAllItems();
5
6     POSITION pos;
7     // ** Parcurgem lista directoarelor selectate în cadrul
8     // ** casetei cu lista
9     for(pos = m_strList.GetHeadPosition(); pos != NULL;)
10    {
11        int nItem;
12        HANDLE hFind;
13        WIN32_FIND_DATA dataFind;
14        BOOL bMoreFiles = TRUE;
15        CString str;
16        CString strFind;
17
18        str = m_strList.GetAt(pos);
19        // ** Adaugăm un rand în controlul lista (coloana 0)
20        nItem = m_lcDirDetails.InsertItem(0, str);
21
22        strFind = m_strMainDir + "\\\" + str + "\\*.\";
23        hFind = FindFirstFile(strFind, &dataFind);
24
25        int nFileCount = 0;
26        double nFileSize = 0;
27
28        // ** Cautăm iterativ toate fișierele din director și
29        // ** însumăm numărul de fișiere și dimensiunile acestora,
30        while(hFind != INVALID_HANDLE_VALUE && ↪❶
31        ↪bMoreFiles == TRUE)
32        {
33            if(dataFind.dwFileAttributes ==
34            ↪FILE_ATTRIBUTE_ARCHIVE)
35            {
36                // ...
37            }
38        }
39    }
```

(continuare)

LISTINGUL 6.7 Continuare

```

34         nFileCount++;
35         nFileSize += (dataFind.nFileSizeHigh *
           ↳ MAXDWORD)
36         + dataFind.nFileSizeLow;
37     }
38     bMoreFiles = FindNextFile(hFind, &dataFind);
39 }
40 // ** Inchidem indicatorul de cautare
41 FindClose(hFind);
42
43 // ** Formatare un sir care contine numarul de
44 // ** fisiere si-l inseram in coloana 1
45 str.Format("%ld", nFileCount); — ③
46 m_lcDirDetails.SetItemText(nItem, 1, str);
47
48 // ** Formatare un sir care contine dimensiunea
49 // ** totala a fisierelor si-l inseram in coloana 2
50 str.Format("%-1.2f", nFileSize / 1024.0); — ④
51 m_lcDirDetails.SetItemText(nItem, 2, str);
52
53 m_strList.GetNext(pos);
54 }
55 }

```

① Inspectam fișierele din fiecare subdirector.

② Incrementăm numărul de fișiere și acumulăm dimensiunea totală a fișierelor.

③ Adăugăm în coloană informația privind numărul de fișiere.

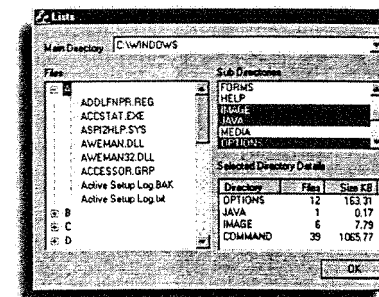
④ Adăugăm în coloană informația privind dimensiunea totală a fișierelor.

Apelul funcției `DeleteAllItems()` din linia 4 înlătură toate elementele existente în cadrul controlului listă, dar nu elimină coloanele. Bucla `for` din linia 9 parcurge iterativ variabila `m_strList` care a fost generată în interiorul funcției `OnSelchangeSubDirs()`. Fiecare obiect `CString` din această listă reprezintă câte un director selectat din caseta cu listă. În apelul `InsertItem()` din linia 20 este transmis numele directorului, iar acesta va reprezenta conținutul coloanei 0. Funcția `InsertItem()` întoarce indicele elementului nou adăugat, considerând zero ca bază de indexare, acest indice fiind reținut în `nItem`. Puteți observa că indicele este transmis apoi în apelurile `SetItemText()` din liniile 46 și 51, efectul fiind stabilirea conținutului pentru celelalte două coloane.

Pentru amănunte privind funcțiile `FindFirstFile()` și `FindNextFile()`, consultați secțiunea „Popularea unui arbore” aflată mai devreme în acest capitol.

Asamblați și rulați programul. Testați selecția multiplă în cadrul casetei cu listă; ar trebui să obțineți rezultate similare cu cele ilustrate în figura 6.7.

FIGURA 6.7
Programul Lists.



VEDEȚI...

➤ Pentru mai multe informații privind stilurile controlului listă, vedeți pagina 427.

CAPITOLUL

7

Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și dată/oră

Utilizarea controalelor indicator de evoluție pentru afișarea evoluției unor operații de durată

Folosirea glisoarelor și a barelor de derulare pentru specificarea unei poziții și a unei selecții într-un anumit interval

Utilizarea noilor controale calendar și de selecție a datei/orei pentru specificarea datelor, a orelor și a intervalelor calendaristice

Controale orientate spre intervale de valori

În programe, va trebui de multe ori să permiteți utilizatorului să specifice o valoare aflată într-un anumit interval de valori sau să definească extremitățile unui interval, sau să oferiți utilizatorului informații despre proprietățile unui interval curent sau să ilustrați evoluția unei operații de durată. Aceste intervale pot reprezenta diferite tipuri de date, fie acestea numerice, calendaristice, orare sau date derivate într-o formă specifică aplicației, precum metrii sau milele. Visual C++ 6 pune la dispoziție o serie de controale, disponibile prin intermediul paletei Controls din cadrul editorului de resurse, împreună cu încapsulări MFC corespunzătoare pentru a vă oferi mecanisme rapide și facile potrivite unor astfel de interacțiuni cu utilizatorul. Capitolul de față se oprește asupra acestor tipuri de controale și prezintă modul în care ele pot fi integrate în aplicații.

Utilizarea unui control indicator de evoluție

În mod normal, controalele indicator de evoluție se folosesc pentru a informa utilizatorul despre desfășurarea unei operații de durată. Un control indicator de evoluție apare sub forma unei bare albastre, continuă sau întreruptă, care umple treptat o casetă pe măsură ce operația înaintază în desfășurare, așa cum se vede în figura 7.1. Nu este obligatoriu să folosiți controalele indicator de evoluție exclusiv pentru a ilustra evoluția unei operații; le puteți utiliza la fel de bine pentru a reprezenta valori dintr-un interval. De pildă, figura 7.2 arată modul în care controalele indicator de evoluție pot fi folosite pentru afișarea unui egalizator grafic, asemănător celor prezente în sistemele audio.

FIGURA 7.1

Un control indicator de evoluție tipic, folosit pentru a ilustra desfășurarea unei operații.

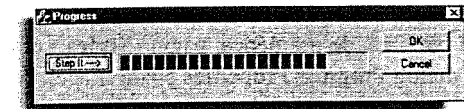
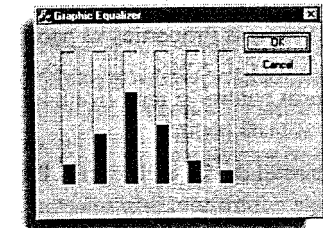


FIGURA 7.2

Controale indicator de evoluție folosite pentru reprezentarea valorilor din intervale, asemeni unui egalizator grafic audio.



Adăugarea unui control indicator de evoluție într-o casetă de dialog

Adăugarea într-o casetă de dialog a unui nou control indicator de evoluție se poate face prin intermediul editorului de resurse, selectând pictograma controlului de pe paleta Controls și poziționând-o pe suprafața casetei de dialog. Dimensiunea controlului poate fi modificată apoi după necesități. Pictograma controlului indicator de evoluție este ilustrată în figura 7.3, aflată sub cursorul mouse-ului și descrisă de o etichetă explicativă.

Copierea și lipirea controalelor

Puteți crea o dublură a unui control existent (oricare ar fi tipul acestuia) din noua casetă de dialog sau din orice altă casetă de dialog folosind operațiile normale de copiere și lipire (Ctrl+C și Ctrl+V). În acest fel va fi creat un nou control care moștenește dimensiunile și proprietățile controlului original pe care l-ați copiat.

FIGURA 7.3

Adăugarea unui control indicator de evoluție într-o casetă de dialog prin intermediul editorului de resurse.

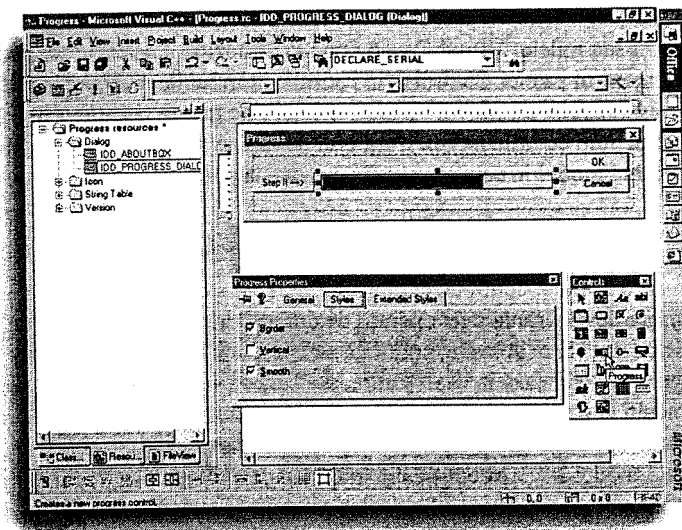


Figura 7.3 prezintă, de asemenea, pagina **Styles** a casetei de dialog **Progress Properties**. Casetă de dialog **Progress Properties** poate fi apelată selectând controlul și apăsând **Alt+Enter** sau efectuând un clic cu butonul drept pe suprafața controlului și selectând opțiunea **Properties** din meniul contextual deschis.

Casetă de dialog **Progress Properties** vă permite modificarea identificatorului unui control în cadrul paginii **General**. Aveți grijă ca identificatorul înscris în câmpul **ID** să fie un nume potrivit pentru control și modificați după nevoie celelalte proprietăți ale controlului care figurează în această pagină.

Pagina **Styles** a casetei de dialog **Progress Properties** vă permite să modificați următoarele trăsături vizuale ale controlului prin selectarea casetelor de validare corespunzătoare:

- **Border**. Dacă această opțiune este selectată, controlul va fi încadrat de un chenar negru, subțire.
- **Vertical**. Dacă această opțiune este selectată, controlul va fi desenat pe verticală, nu pe orizontală.
- **Smooth**. Dacă această opțiune este selectată, bara care indică evoluția va fi continuă, și nu formată din blocuri.

Stilul implicit al indicatorului de evoluție

Dacă nu efectuați modificări în cadrul paginii **Styles** corespunzătoare controlului indicator de evoluție, în mod implicit stilul **Border** este activ, iar stilurile **Vertical** și **Smooth** sunt dezactivate.

Exemplul înfățișat în figura 7.1 arată cum o aplicație simplă de tip dialog, generată cu AppWizard, poate fi dezvoltată pentru a afișa un indicator de evoluție care este incrementat la apăsarea butonului **Step It-->**. Dacă doriți să urmați acest exemplu, creați aplicația de tip dialog cu ajutorul AppWizard și adăugați un indicator de evoluție și un buton, ca în figura 7.3. Controlul indicator de evoluție din exemplu are identificatorul **IDC_MY_PROGRESS**, iar butonul are identificatorul **IDC_STEPIT**. Secțiunile care urmează arată cum se poate atașa o clasă MFC la un control indicator de evoluție și cum poate fi actualizată bara afișată de către control.

VEDEȚI...

➤ Pentru detalii legate de editarea casetelor de dialog și de proprietățile controalelor, vedeți pagina 54.

Maparea unei variabile peste un control indicator de evoluție

Pentru a mapa o variabilă peste un control indicator de evoluție puteți apela la ClassWizard. Controalele indicator de evoluție sunt mapate prin intermediul clasei MFC **CProgressCtrl**.

Maparea unei variabile membru peste un control indicator de evoluție

1. În cadrul editorului de resurse, selectați noul control indicator de evoluție pentru care doriți maparea efectuând un clic pe acesta.
2. Apăsați **Ctrl+W** sau efectuați un clic pe meniul **View** și selectați opțiunea **ClassWizard** pentru a deschide ClassWizard și a accesa pagina **Member Variables** a acestuia. Asigurați-vă că în caseta **Class Name** este înscrisă clasa corespunzătoare noului control (de exemplu, **CProgressDlg**).
3. Puteți fie să efectuați un dublu clic pe identificatorul corespunzător noului control indicator de evoluție, fie să efectuați un clic pe **Add Variable** pentru a apela caseta de dialog **Add Member Variable** asociată controlului.

Controlul standard indicator de evoluție

Deoarece controlul indicator de evoluție este unul dintre noile controale standard, pentru a putea utiliza acest control și alte controale standard este necesar Windows 95/NT 3.51 sau o versiune mai recentă de sistem de operare.

4. Introduceți în caseta de editare **Member Variable Name** un nume pentru noul membru (cum ar fi **m_MyProgress**, pentru programul nostru).
5. Observați că în caseta combinată **Category** este înscris **Control**, iar câmpul **Variable Type** conține **CProgressCtrl**. Acestea sunt singurele opțiuni disponibile pentru controlul indicator de evoluție.

6. Efectuați un clic pe **OK** pentru a confirma maparea noului control și închideți caseta de dialog **Add Member Variable**. Numele noii variabile membru pe care ați mapat-o ar trebui să apară în partea dreaptă a identificatorului asociat controlului, în cadrul listei **Control ID**.
7. Efectuați un clic pe **OK** pentru a închide **ClassWizard** și a adăuga noua variabilă membru în clasa dorită.

Odată efectuată maparea controlului, puteți opera asupra variabilei membru mapate pentru a modifica proprietățile vizuale ale controlului, așa cum o va arăta secțiunea următoare.

VEDEȚI...

➤ Pentru explicații amănunțite privind maparea variabilelor membru, vedeți pagina 206.

Manipularea și actualizarea controlului indicator de evoluție

Controalele indicator de evoluție pot fi manipulate prin apeluri de metode ale variabilelor membru `CProgressCtrl` mapate. Controlul indicator de evoluție are asociat un interval care definește valorile întregi corespunzătoare stărilor controlului, de la un control gol (umplut 0%) până la un control plin (umplut 100%). Există, de asemenea, o poziție curentă ce determină procentul în care trebuie umplut controlul, în corespondență cu avansul în cadrul intervalului definit. În fine, puteți stabili o valoare de increment cu care este avansată poziția controlului la fiecare apel al funcției membru `StepIt()`.

Stabilirea intervalului pentru controlul indicator de evoluție

Înainte de toate ar trebui să stabiliți intervalul pentru controlul indicator de evoluție. Aceste valori pot fi stabilite apelând funcția membru `SetRange()` a controlului, transmitând două valori întregi reprezentând limitele inferioară și superioară ale intervalului. De regulă veți fixa intervale care corespund în mod direct operației pe care o efectuați. De exemplu, în cazul în care calculați numerele prime între 3000 și 7000, puteți stabili ca margini ale intervalului tocmai aceste numere.

Utilizarea intervalelor pe 32 de biți în cadrul indicatoarelor de evoluție

Funcția `SetRange()` este limitată de faptul că acceptă numai valori pe 16 biți. Aceasta înseamnă că intervalul maxim care poate fi stabilit este cuprins între -32768 și 32768. Dacă aveți nevoie de un interval mai întins sau de limite mai mari, va trebui să apelați la funcția `SetRange32()`, care acceptă valori pe 32 de biți. Astfel puteți stabili limite între -2147483648 și 2147483647.

Probabil că veți fixa intervalul o singură dată, la inițializarea controlului (deși îl puteți stabili din nou în orice moment). Un loc evident pentru inițializarea controlului este sfârșitul funcției `OnInitDialog()` a casetei de dialog.

În exemplul nostru puteți stabili intervalul controlului între 0 și 10. Atunci când poziția curentă este 0 controlul apare gol, iar când poziția devine 10 controlul este afișat plin. Dacă efectuați un clic pe eticheta paginii **ClassView** a secțiunii spațiului de lucru al proiectului și apoi efectuați clic pe semnele plus din dreptul numelui proiectului și al

claselor, veți putea vizualiza clasele din proiect și membrii acestora. Dacă numele aplicației de tip dialog este **Progress**, caseta de dialog principală a programului va avea asociată clasa corespunzătoare `CProgressDlg`. În cadrul acesteia ar trebui să figureze o funcție membru `OnInitDialog()`, pe care o puteți vizualiza într-o fereastră de editare prin simpla efectuare a unui dublu clic pe numele său afișat în secțiunea **ClassView**. Pentru inițializarea intervalului asociat indicatorului de evoluție, puteți adăuga următoarea linie de cod la finalul funcției `OnInitDialog()`, chiar înainte de instrucțiunea de revenire:

```
m_MyProgress.SetRange(0,10);
```

De asemenea, există o funcție inversă `GetRange()` care primește doi parametri întregi prin referință și înscrie în primul limita inferioară a intervalului, iar în al doilea limita superioară. Puteți folosi această funcție pentru a determina intervalul curent stabilit pentru un control indicator de evoluție.

VEDEȚI...

➤ Pentru mai multe informații privind inițializarea casetelor de dialog, vedeți pagina 121.

Stabilirea poziției pentru controlul indicator de evoluție

Odată fixat intervalul indicatorului de evoluție, puteți actualiza poziția afișată prin apelul funcției `SetPos()`. Aceasta va actualiza poziția controlului la valoarea întreagă transmisă în apel și va duce la redesenarea controlului pentru a reflecta modificarea survenită. Dacă valoarea transmisă depășește limita superioară a intervalului, bara desenată va umple întreg controlul, iar dacă valoarea coboară sub limita inferioară a intervalului, controlul va fi desenat gol.

Aplicația exemplificativă Fire

Aplicația exemplificativă **Fire** este o demonstrație spectaculoasă a controalelor indicator de evoluție și glisor. Acest exemplu poate fi găsit pe CD-ul MSDN sau pe situl de Web al MSDN, la adresa www.microsoft.com/msdn, unde veți căuta cuvântul **FIRE**.

În programul nostru puteți adăuga o linie care stabilește poziția inițială a controlului la 0 imediat după fixarea intervalului de valori, ca mai jos:

```
m_MyProgress.SetPos(0);
```

Această inițializare nu este neapărat necesară, pentru că poziția controlului este inițializată implicit la limita inferioară a intervalului. Ați putea apoi să afișați controlul pe jumătate plin adăugând linia următoare:

```
m_MyProgress.SetPos(5);
```

În loc să actualizați controlul cu o valoare absolută aflată în interiorul intervalului fixat, puteți modifica poziția curentă cu o cantitate relativă prin intermediul funcției `OffsetPos()`. Atunci când transmiteți o valoare funcției `OffsetPos()`, aceasta este adunată la poziția curentă și controlul este redesenat pentru a reflecta modificarea survenită.

Fixarea și utilizarea valorii de increment

Puteți să stabiliți o valoare de increment cu care controlul să fie actualizat automat la primirea unui semnal de avans. Funcția `SetStep()` este cea care vă permite fixarea unei astfel de valori de increment, transmisă în apel ca o valoare întreagă. Actualizarea poziției controlului indicator de evoluție se poate efectua ulterior printr-un semnal transmis de apelul funcției `StepIt()` fără parametri.

Spre exemplu, ați putea adăuga în program o nouă linie la sfârșitul funcției `OnInitialUpdate()` pentru a stabili valoarea de increment a controlului la 1, ca mai jos:

```
m_MyProgress.SetStep(1);
```

Mai departe, ați putea adăuga o funcție de răspuns pentru butonul **Step It-->** de pe caseta de dialog (efectuând un dublu clic pe buton în cadrul editorului de resurse), funcție care să apeleze metoda `StepIt()` a indicatorului de evoluție ca mai jos:

```
void CProgressDlg::OnStepIt()
{
    m_MyProgress.StepIt();
}
```

Dacă asamblați și rulați programul după adăugarea acestui cod de răspuns la apăsarea butonului, veți vedea că efectuarea unui clic pe butonul **Step It-->** are ca efect avansul controlului indicator de evoluție. Veți observa, de asemenea, că odată ajuns la lungimea maximă a barei după 10 clicuri, un nou apel `StepIt()` goleşte controlul și procesul se repetă.

Utilizarea unui control bară de derulare

Barele de derulare se întâlnesc de multe ori atașate de marginile unei ferestre în scopul derulării conținutului acestora (așa cum se descrie în amănunt în capitolul 18, „Dimensionarea și derularea reprezentărilor”). Cu toate acestea, ele pot fi utilizate la fel de bine independent pentru a specifica o poziție în cadrul unui interval precizat. În prezent, acest rol a fost preluat în bună măsură de controlul glisor (prezentat în secțiunea următoare); chiar și așa, este bine să aflați despre barele de derulare deoarece controlul glisor împrumută mult din funcționalitatea acestora.

VEDEȚI...

➤ Pentru mai multe informații privind utilizarea barelor de derulare la derularea unei ferestre, vedeți pagina 415.

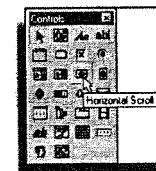
Adăugarea unui control bară de derulare într-o casetă de dialog

Editorul de resurse de tip dialog vă permite preluarea unui control bară de derulare din paleta Controls și plasarea acestuia pe suprafața casetei de dialog. În paletă există două pictograme asociate barelor de derulare, una corespunzătoare barelor de derulare orizontale (evidențiată în figura 7.4 de o etichetă explicativă) și cealaltă, alăturată, corespunzătoare barelor de derulare verticale. Puteți apoi să poziționați și să dimensionați barele de derulare după nevoie și să

stabiliți identificatori corespunzători în cadrul paginii **General** a casetei de dialog Scrollbar Properties (apelată prin apăsarea combinației **Alt+Enter** după selectarea controlului).

FIGURA 7.4

Adăugarea barelor de derulare verticale și orizontale în cadrul editorului de resurse de tip dialog.



Pagina **Styles** a casetei de dialog Scrollbar Properties conține o singură opțiune care vă permite stabilirea alinierii barei de derulare, opțiune numită **Align**. Această aliniere poate avea una din următoarele trei valori:

- **None.** Aceasta este valoarea implicită și va duce la desenarea unei bare de derulare având aceleași dimensiuni cu bara de derulare plasată pe macheta casetei de dialog.
- **Top/Left.** Această opțiune va duce la desenarea unei bare de derulare având lățimea standard și fiind aliniată la marginile din stânga și superioară ale barei de derulare plasate pe macheta casetei de dialog.
- **Bottom/Right.** Această opțiune are același efect cu opțiunea **Top/Left**, dar bara de derulare va fi aliniată la marginile din dreapta și inferioară ale barei de derulare plasate pe macheta casetei de dialog.

VEDEȚI...

➤ Pentru mai multe detalii privind editarea casetelor de dialog și proprietățile controalelor, vedeți pagina 54.

Maparea unei variabile peste un control bară de derulare

Maparea unei variabile peste o bară de derulare se poate face urmând secvența de pași „Maparea unei variabile membru peste un control indicator de evoluție”, prezentată mai devreme în acest capitol. Singura diferență notabilă ține de caseta combinată **Category** și de câmpul **Variable Type**, menționate în pasul 5. Un control bară de derulare nu permite decât opțiunea **Control** în caseta **Category**, în timp ce un control glisor vă permite să alegeți aici între opțiunile **Control** și **Value**.

Dacă selectați opțiunea **Value**, câmpul **Variable Type** este completat cu `int`. În consecință, în clasa destinație pe care ați selectat-o va fi adăugată o variabilă membru de tip întreg, mapată peste controlul glisor. Această nouă variabilă va fi apoi actualizată de poziția controlului într-o manieră similară cu actualizarea unui întreg de către un control de editare (aspect discutat în detaliu în secțiunea „Adăugarea de variabile membru pentru stocarea datelor din caseta de dialog” din capitolul 10, „Utilizarea casetelor de dialog”).

În cazul în care selectați opțiunea **Control**, câmpul **Variable Type** va fi completat automat cu clasa MFC corespunzătoare, `CScrollBar`. Puteți continua secvența de pași „Maparea unei variabile membru peste un control indicator de evoluție” pentru a închide caseta de dialog **Add Member Variable** și **Class Wizard**.

Utilizarea opțiunii Control la mapare

De fiecare dată când mapați un control folosind opțiunea **Control**, clasa cu care se realizează maparea (de exemplu, **CScrollBar**) va fi întotdeauna derivată din clasa **CWnd**. Aceasta înseamnă că orice operație pe care o puteți efectua asupra unei ferestre, cum ar fi modificarea dimensiunii și a poziției, poate fi efectuată în cod prin intermediul variabilei membru care este mapată. Puteți vedea un exemplu de control de editare redimensionat astfel în secțiunea „Tratarea evenimentului de fixare a dimensiunii” din capitolul 18, „Dimensionarea și derularea reprezentărilor”.

VEDEȚI...

► Pentru explicații detaliate privind maparea variabilelor membru, vedeți pagina 206.

Inițializarea unui control bară de derulare

Puteți inițializa o bară de derulare în cadrul funcției **OnInitDialog()**, așa cum ați procedat în cazul controlului indicator de evoluție. Bara de derulare are asociată și ea un interval, asemeni indicatorului de evoluție, interval ce poate fi stabilit apelând funcția **SetScrollRange()** a clasei **CScrollBar** și transmitând în apel limitele inferioară și superioară ca parametru prim și, respectiv, secund. Mai este posibilă precizarea unui indicator de redesenare opțional ca un al treilea parametru (indicator care, omis, are implicit valoarea **TRUE**).

De exemplu, dacă mapați peste barele de derulare ilustrate în figura 7.4 două variabile membru de tip **CScrollBar**, numite **m_ScrollBar1** (pentru bara verticală) și **m_ScrollBar2** (pentru bara orizontală), puteți inițializa ambele controale la sfârșitul funcției **OnInitDialog()** prin intermediul liniilor de cod următoare:

```
m_ScrollBar1.SetScrollRange(0,100);
m_ScrollBar2.SetScrollRange(0,200);
```

Limitele intervalelor asociate cu barele de derulare

Puteți transmite orice valori întregi ca limite inferioară și superioară pentru interval, dar diferența dintre aceste două valori nu trebuie să depășească 32767.

În acest caz, bara de derulare verticală (**m_ScrollBar1**) va avea asociat un interval de la 0 la 100, în timp ce bara de derulare orizontală (**m_ScrollBar2**) va avea limita inferioară 0 și limita superioară 200. Nu a fost transmis un al treilea parametru, ceea ce înseamnă că ambele bare sunt redesenate implicit (ați putea transmite valoarea **FALSE** ca al treilea parametru pentru a inhiba redesenarea). Există o funcție inversă **GetScrollRange()** căreia îi puteți transmite doi pointeri la întregi în care veți obține limitele intervalului de derulare curent, ca aici:

```
int nMin, nMax;
m_ScrollBar2.GetScrollRange(&nMin, &nMax);
TRACE("Range = (%d to %d)\n", nMin, nMax);
```

Dacă doriți să dezactivați vreuna din săgețile desenate la capetele unei bare de derulare, puteți apela funcția **EnableScrollBar()** a clasei corespunzătoare, transmitând una din valorile simbolice prezentate în tabelul 7.1.

TABELUL 7.1 Valori simbolice pentru dezactivarea săgeților barelor de derulare prin apelul **EnableScrollBar()**

Valoare simbolică	Descriere
ESB_DISABLE_BOTH	Dezactivează ambele butoane de pe bara de derulare.
ESB_DISABLE_LTUP	Dezactivează butonul din stânga sau superior (în funcție de orientarea barei de derulare).
ESB_DISABLE_RTDN	Dezactivează butonul din dreapta sau inferior (în funcție de orientarea barei de derulare).
ESB_ENABLE_BOTH	Activează ambele butoane ale barei de derulare (aceasta este acțiunea implicită în cazul în care nu este transmis nici un parametru în apelul EnableScrollBar()).

Asemeni controlului indicator de evoluție, puteți stabili poziția curentă a barei de derulare, ilustrată de caseta mobilă aflată pe bară, apelând funcția **SetScrollPos()** și transmitând o valoare întreagă ce reprezintă poziția în cadrul intervalului definit. Există și o funcție inversă, **GetScrollPos()**, care întoarce poziția curentă.

Dacă doriți ca dimensiunea casetei de derulare să reprezinte un anumit procent din intervalul definit (cum ar fi dimensiunea unei pagini față de întregul document), puteți apela la funcția **SetScrollInfo()**, transmitând un pointer la o structură **SCROLLINFO**. Cea mai mare parte a acestei structuri dublează funcțiile de stabilire a poziției și a intervalului despre care am discutat mai devreme.

Structura SCROLLINFO

GetScrollInfo() este funcția pereche pentru **SetScrollInfo()** și poate fi utilizată pentru a completa o structură **SCROLLINFO** cu informații privind o bară de derulare. **GetScrollInfo()** necesită doi parametri, primul fiind un pointer la structura **SCROLLINFO** pe care doriți să o completați, iar al doilea fiind o mască de biți compusă din indicatori corespunzători valorilor **SCROLLINFO** care vă interesează. Această mască poate fi o combinație între indicatorii **SIF_RANGE**, **SIF_POS**, **SIF_TRACKPOS** și **SIF_PAGE**, corespunzători respectiv variabilelor membru **nMin** și **nMax**, **nPos**, **nTrackPos** și **nPage** ale structurii **SCROLLINFO**. Ca alternativă, indicatorul **SIF_ALL** extrage toate valorile.

Variabila membru a structurii **SCROLLINFO** care ne interesează este **nPage**. Va trebui să atribuiți acestei variabile o valoare întreagă care reprezintă dimensiunea paginii vizibile în raport cu întregul interval de derulare. Spre exemplu, să presupunem că dezvoltăți o aplicație de prelucrare a textelor. Intervalul asociat barei de derulare a aplicației pentru întregul document ar putea fi între 0 și 100, iar pe ecran s-ar putea să afișați o singură pagină la un moment dat. Așadar, pentru un document de două pagini veți vrea să atribuiți variabilei membru **nPage** valoarea 50, pentru a arăta că o pagină reprezintă jumătate din întregul document. Pentru un document de 20 de pagini, în schimb, ar trebui să atribuiți variabilei membru **nPage** valoarea 5, pentru a arăta că o pagină reprezintă a 20-a parte din întregul document (adică a 20-a parte din intervalul de la 0 la 100).

Trebuie, în plus, să atribuieți membrului `fMask` valoarea `SIF_PAGE` pentru a informa funcția `SetScrollInfo()` că membrul `nPage` este cel semnificativ și trebuie să înregistrați în membrul `cbSize` dimensiunea structurii (situație uzuală în cazul structurilor Win32). De exemplu, pentru a stabili dimensiunea unei pagini pentru o bară de derulare verticală la 30, în codul de inițializare ar trebui să adăugați următoarele linii după apelul

```
SetScrollRange();

SCROLLINFO si;
si.cbSize = sizeof(SCROLLINFO);
si.nPage = 30;
si.fMask = SIF_PAGE;
m_ScrollBar1.SetScrollInfo(&si);
```

După execuția acestor linii, dimensiunea casetei de derulare a barei verticale va fi mai mare decât dimensiunea implicită a casetei aflate pe bara de derulare orizontală.

VEDEȚI...

➤ Pentru mai multe informații privind macrodefiniția `TRACE`, vedeți pagina 639.

Tratarea mesajelor trimise de barele de derulare

De fiecare dată când utilizatorul modifică poziția barei de derulare, apasă unul dintre butoanele acesteia sau apasă tastele Page Up/Page Down sau săgețile sus/jos, bara de derulare în cauză trimite un mesaj corespunzător ferestrei care o conține (fereastra părinte). Aceste mesaje sunt mesajele Windows `WM_HSCROLL` și `WM_VSCROLL`, corespunzătoare barelor de derulare orizontale și, respectiv, verticale. Crearea de funcții de tratare pentru aceste mesaje se face în maniera cunoscută, prin intermediul casetei de dialog **New Windows Messages/Events**. Clasa dialog principală a unei aplicații de tip dialog este cea care încapsulează fereastra părinte a barei de derulare, prin urmare funcția de tratare a mesajului trebuie creată aici. De pildă, dacă aplicația de tip dialog se numește `Scroll`, clasa principală a sa va fi `CScrollDlg`; aceasta este clasa care trebuie selectată în lista **Class or Object to Handle** din caseta de dialog **New Windows Messages/Events**. (Dacă aveți nevoie de ajutor în acest punct, puteți să urmați secvența de pași „Adăugarea unei funcții de tratare `OnHScroll()` care să răspundă la mesajul `WM_HSCROLL`” din capitolul 18, „Dimensionarea și derularea reprezentărilor”. Va trebui să înlocuiți referirile la clasa reprezentare cu clasa dialog principală și să urmăriți mesajul `WM_VSCROLL`, nu `WM_HSCROLL`.)

După inserarea noii funcții pentru tratarea mesajului `WM_VSCROLL`, veți putea vedea următoarea secvență de cod generată de către **ClassWizard**:

```
void CScrollDlg::OnVScroll(UINT nSBCode, UINT nPos,
    CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call
    CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

Funcția de tratare a mesajelor `WM_HSCROLL` este identică atât ca definiție cât și ca implementare, cu deosebirea totuși că este apelată `OnHScroll()`. Dacă aveți atât bare de derulare orizontale cât și verticale, veți avea nevoie de funcții de tratare separate pentru cele două orientări.

Poziții negative pe bara de derulare

Dacă intervalul fixat pentru bara de derulare permite reprezentarea numerelor negative, este posibil ca `nPos` să conțină o valoare negativă. Dar pentru că se transmite ca o valoare întreagă fără semn (`UINT`), valoarea negativă va fi reprezentată printr-un număr pozitiv mare. Dacă știți că așa stau lucrurile la un moment dat, convertiți variabila `nPos` la tipul `int` pentru a putea opera cu valorile negative.

Primul parametru transmis (`nSBCode`) este un indicator care arată tipul operației de derulare efectuate de către utilizator. Această operație depinde de modul în care a fost utilizat controlul bară de derulare (de exemplu, a fost trasă caseta de derulare, s-a efectuat clic pe una din săgeți sau în interiorul barei sau a fost apăsată o tastă săgeată sau de pagină). Valorile posibile pentru acest indicator sunt enumerate în tabelul 7.2. Mai întâi trebuie să decideți dacă doriți să răspundeți la bara de derulare (în cazul în care este o bară de derulare a unei ferestre, situație în care funcția de tratare `OnVScroll()` a clasei de bază `CDialog` se va ocupa de aceasta). Apoi puteți folosi valoarea indicatorului menționat pentru a decide modul în care ar trebui actualizată bara de derulare. Dacă nu modificați explicit poziția barei de derulare printr-un apel al funcției `SetScrollPos()` asociate, în momentul în care utilizatorul eliberează caseta de derulare aceasta va sări înapoi la capătul controlului!

TABELUL 7.2 Valorile indicatorului transmis prin parametru `nSBCode` al unei funcții care răspunde la bara de derulare

Valoarea indicatorului	Semnificație
<code>SB_THUMBTRACK</code>	Utilizatorul a tras caseta de derulare și a ajuns într-o anumită poziție. Această poziție este dată de al doilea parametru, <code>nPos</code> .
<code>SB_THUMBPOSITION</code>	Utilizatorul a tras caseta de derulare până într-o anumită poziție și a eliberat butonul mouse-ului. Această poziție este dată de al doilea parametru, <code>nPos</code> .
<code>SB_ENDSCROLL</code>	Utilizatorul a eliberat butonul mouse-ului după ce l-a ținut apăsat deasupra uneia dintre săgețile de la capete sau în interiorul barei (nu deasupra casetei de derulare).
<code>SB_LINEUP</code>	Poziția casetei de derulare trebuie decrementată cu unu.
<code>SB_LINELEFT</code>	Similar cu <code>SB_LINEUP</code> , dar pentru barele de derulare orizontale.
<code>SB_LINEDOWN</code>	Poziția casetei de derulare trebuie incrementată cu unu.
<code>SB_LINERIGHT</code>	Similar cu <code>SB_LINEDOWN</code> , dar pentru barele de derulare orizontale.
<code>SB_PAGEUP</code>	Poziția casetei de derulare trebuie decrementată cu dimensiunea unei pagini (specifică aplicației).
<code>SB_PAGELLEFT</code>	Similar cu <code>SB_PAGEUP</code> , dar pentru barele de derulare orizontale.
<code>SB_PAGEDOWN</code>	Poziția casetei de derulare trebuie incrementată cu dimensiunea unei pagini (specifică aplicației).
<code>SB_PAGERIGHT</code>	Similar cu <code>SB_PAGEDOWN</code> , dar pentru barele de derulare orizontale.

Puteți afla ce bară de derulare a trimis mesajul inspectând cel de-al treilea parametru trimis funcției de tratare `OnVScroll()`, acesta fiind un pointer către bara de derulare asupra căreia a acționat utilizatorul. Probabil că cea mai bună și mai simplă cale de a identifica bara de derulare este să folosiți identificatorul acesteia. Identificatorul poate fi aflat apelând funcția `GetDlgCtrlID()` pentru pointerul `pScrollBar`. Această funcție va întoarce identificatorul asociat controlului, pe care îl puteți compara apoi cu controalele cunoscute pentru a decide dacă (și cum) veți răspunde la mesaj.

Mesajele `WM_VSCROLL` și `WM_HSCROLL`

Trebuie să rețineți că barele de derulare verticale trimit mesajul `WM_VSCROLL`, care este tratat de funcția `OnVScroll()`. În schimb, barele de derulare orizontale trimit mesajul `WM_HSCROLL` și necesită o altă funcție de tratare, `OnHScroll()`. O greșeală frecventă este implementarea unei singure funcții de tratare, care răspunde exclusiv fie la mesajele trimise de barele de derulare orizontale, fie la cele trimise de barele de derulare verticale. Poate că era mai bine dacă Microsoft ar fi optat pentru trimiterea unui singur mesaj pentru toate barele de derulare, împreună cu un indicator al orientării. Din păcate, însă, limitările originale impuse de operarea pe 16 biți din zilele timpurii ale Windows au condus la necesitatea existenței a două mesaje.

Codul prezentat în listingul 7.1 reprezintă un exemplu de tratare a mesajului `WM_VSCROLL`. După ce se identifică bara de derulare, se determină tipul operației și se iau măsurile adecvate pentru actualizarea poziției în cadrul barei de derulare. În continuare, poziția barei de derulare orizontală este de asemenea actualizată pentru a oglindi modificările aduse barei verticale, ceea ce înseamnă că dacă utilizatorul trage caseta de derulare a barei verticale, caseta barei orizontale se va deplasa și ea.

LISTINGUL 7.1 LST7_1.CPP – Tratarea mesajelor trimise de barele de derulare și actualizarea poziției acestora

```
1 void CScrollDlg::OnVScroll(UINT nSBCode, UINT nPos,
   CScrollBar* pScrollBar)
2 {
3     if (pScrollBar->GetDlgCtrlID() == IDC_SCROLLBAR1) — ❶
4     {
5         int nCurrentPos = pScrollBar->GetScrollPos();
6         switch(nSBCode)
7         {
8             case SB_THUMBTRACK:
9             case SB_THUMBPOSITION:
10                pScrollBar->SetScrollPos(nPos); — ❷
11                break;
12            case SB_LINEUP:
13                pScrollBar->SetScrollPos(nCurrentPos-1);
14                break;
15            case SB_LINEDOWN:
16                pScrollBar->SetScrollPos(nCurrentPos+1);
17                break;
18            case SB_PAGEUP:
```

```
19                pScrollBar->SetScrollPos(nCurrentPos-5);
20                break;
21            case SB_PAGEDOWN:
22                pScrollBar->SetScrollPos(nCurrentPos+5);
23                break;
24        }
25        m_ScrollBar2.SetScrollPos( — ❸
26            2 * pScrollBar->GetScrollPos());
27    }
28
29    CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
30 }
```

- ❶ Verificăm identificatorul pentru a ne asigura că mesajul este destinat barei de derulare potrivite.
- ❷ Actualizăm poziția casetei de derulare în funcție de mesajul trimis de bara de derulare în urma interacțiunii cu utilizatorul.
- ❸ Această linie actualizează poziția celei de-a doua bare de derulare la dublul valorii primei, deoarece intervalul de derulare pentru a doua bară este dublu față de intervalul pentru prima bară.

Linia 3 din listingul 7.1 determină bara de derulare în cauză, verificând dacă identificatorul întors de `GetDlgCtrlID()` este același cu cel al barei de derulare verticale (`IDC_SCROLLBAR1`). Dacă bara de derulare este cea potrivită, în linia 5 se determină poziția curentă prin apelul funcției `GetScrollPos()` și această poziție este atribuită variabilei `nCurrentPos`.

Instrucțiunea `switch` din linia 6 face distincția între diferitele valori ale parametrului `nSBCode` corespunzătoare acțiunilor utilizatorului. Dacă are loc tragerea casetei de derulare sau dacă aceasta a fost trasă și eliberată, este suficientă stabilirea poziției barei de derulare la poziția `nPos` solicitată de utilizator, în acest scop fiind apelată funcția `SetScrollPos()` în linia 10.

Dacă bara de derulare este deplasată în sus sau în jos cu o linie (prin efectuarea unui clic pe una din săgețile de la capete), poziția curentă este incrementată sau decrementată cu o unitate, toate acestea în liniile de la 12 la 17.

Dacă utilizatorul efectuează un clic în interiorul barei de derulare pentru a solicita deplasarea cu o pagină, poziția curentă a barei de derulare este incrementată sau decrementată cu un număr arbitrar de cinci unități care reprezintă lungimea unei pagini, toate acestea în liniile de la 18 la 23.

În fine, cealaltă bară de derulare (orizentală) peste care este mapată variabila `m_ScrollBar2` este făcută să reflecte poziția barei verticale prin apelul funcției `SetScrollPos()` asociate, în apel fiind transmisă poziția primei bare, acestea petrecându-se în liniile 25 și 26. Observați în linia 26 că poziția primei bare este înmulțită cu 2. Motivul este faptul că intervalul pentru bara de derulare orizentală este dublu față de intervalul pentru bara de derulare verticală, fiind fixat în `OnInitDialog()` prin apelul `SetScrollRange(0,200)`.

Apelul funcției de tratare `CDialog::OnVScroll()` din clasa de bază a casetei de dialog, în linia 29, permite casetei de dialog să răspundă barelor de derulare pe care le posedă prin derularea propriei ferestre (despre aceasta vom discuta mai amănunțit în secțiunea „Derularea ferestrelor” din capitolul 18, „Dimensionarea și derularea reprezentărilor”).

VEDEȚI...

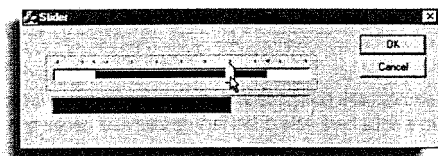
➤ Pentru mai multe informații privind mesajele trimise de barele de derulare, vedeți pagina 422.

Utilizarea unui control glisor

Un control glisor permite utilizatorului să stabilească o valoare prin tragerea unui indicator de-a lungul unui interval liniar și plasarea acestuia într-o anumită poziție (așa cum se vede în figura 7.5), asemănător unui potențiomtru de volum al unui sistem audio.

FIGURA 7.5

Un control glisor cu diviziuni (sus); poziția este indicată de un control indicator de evoluție (jos).



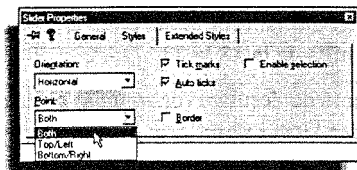
Controlul glisor este un echivalent sofisticat al controlului bară de derulare prin aceea că are asociate de asemenea un interval ce poate fi definit și o poziție curentă. Cu toate acestea, controlul glisor permite o personalizare și o configurare mult mai complexe decât controlul bară de derulare.

Adăugarea unui control glisor într-o casetă de dialog

Puteți plasa un control glisor pe suprafața unei casete de dialog după selectarea sa din cadrul paletelor Controls, așa cum ați procedat și în cazul altor controale (pictograma controlului glisor din paleta de controale se află în dreapta pictogramei indicatorului de evoluție). Ca de obicei, puteți să dimensionați și să poziționați controlul după nevoie și este bine să stabiliți un identificator adecvat pentru control prin intermediul paginii **General** a casetei de dialog **Slider Properties**. Pagina **Styles** din caseta de dialog **Slider Properties** (înfățișată în figura 7.6) vă permite să stabiliți mai multe elemente privind comportamentul controlului și aspectul său vizual.

FIGURA 7.6

Stabilirea proprietăților de stil ale controlului glisor în cadrul editorului de resurse.



În pagina **Styles** se află două casete combinate. Prima dintre ele este **Orientation**, care permite alegerea uneia dintre opțiunile **Horizontal** și **Vertical**. Glisorul este implicit orizontal; dacă îl transformați în glisor vertical, s-ar putea să fiți nevoit să redimensionați

ferestra controlului pentru a o adapta noului aspect. Cea de-a doua casetă combinată vă permite să alegeți un tip de cursor. În mod normal este selectat tipul **Both**, care are ca efect desenarea unei casete rectangulare pe post de indicator de poziție. Puteți înlocui tipul **Both** cu **Top/Left**, ceea ce va duce la afișarea unui indicator al cărui vârf este îndreptat către partea superioară a unui glisor orizontal, respectiv către dreapta unui glisor vertical. Ca alternativă, puteți selecta tipul **Bottom/Right** pentru afișarea unui indicator al cărui vârf este îndreptat în partea opusă față de indicatorul **Top/Left**.

Pot fi selectate și o serie de casete de validare, semnificațiile acestora fiind descrise în continuare:

- **Tick Marks.** Dacă această opțiune este selectată, glisorul va afișa linii mici (diviziuni), perpendiculare pe direcția controlului și situate în partea indicată de cursor, scopul lor fiind să ajute utilizatorul să aprecieze mai precis poziția curentă a glisorului.
- **Auto Ticks.** Dacă această opțiune este selectată, diviziunile sunt plasate de-a lungul intervalului astfel încât să corespundă valorii de increment curente.
- **Enable Selection.** Dacă această opțiune este selectată, este adăugată o bară albă care permite programului să afișeze un interval de selecție prin intermediul unor mici triunghiuri.
- **Border.** Dacă această opțiune este selectată, în jurul controlului glisor este trasat un cadru negru, subțire.

VEDEȚI...

➤ Pentru mai multe amănunte despre editarea casetelor de dialog și proprietățile controalelor, vedeți pagina 54.

Maparea unei variabile peste un control glisor

Operația de mapare a unei variabile peste un control glisor este foarte asemănătoare cu cea corespunzătoare unui control indicator de evoluție. Puteți să urmăriți secvența de pași „Maparea unei variabile membru peste un control indicator de evoluție”, prezentată mai devreme în acest capitol; singura diferență notabilă privește caseta combinată **Category** și câmpul **Variable Type**, menționate în pasul 5. Ca și în cazul barelor de derulare, controlul glisor permite selectarea uneia dintre opțiunile **Control** sau **Value** în cadrul casetei combinate **Category**.

Activarea și dezactivarea dinamică a controalelor

Puteți folosi noua variabilă control mapată (precum `CSliderCtrl`) pentru activarea sau dezactivarea prin program a controlului respectiv. Aceasta se poate face apelând funcția `EnableWindow()` a variabilei mapate, transmitând `TRUE` sau `FALSE` pentru a activa sau, respectiv, dezactiva fereastra (de exemplu, `m_mySliderCtrl.EnableWindow(FALSE)`). Acest lucru l-ați putea dori pentru a împiedica utilizatorul să modifice starea controlului în anumite circumstanțe.

Dacă selectați opțiunea **Value**, câmpul **Variable Type** este completat cu `int`. Aceasta va duce la crearea în clasa destinație selectată a unei variabile membru de tip întreg, mapată

pe controlul glisor. Noua variabilă mapată va fi apoi actualizată de poziția controlului, asemănător cu cazul barei de derulare peste care este mapat un întreg.

Restul acestei secțiuni se va concentra asupra opțiunii **Control**, care duce la înscrierea automată în câmpul **Variable Type** a clasei MFC `CSliderCtrl` pentru maparea controalelor glisor. Odată stabilite detaliile privind maparea variabilei, puteți continua normal cu pașii din secvența „Maparea unei variabile membru peste un control indicator de evoluție”.

VEDEȚI...

► Pentru o explicație mai detaliată privind maparea variabilelor membru, vedeți pagina 206.

Inițializarea unui control glisor

Intervalul unui control glisor poate fi stabilit prin apelul funcției `SetRange()` asociate, în apel transmițându-se ca primi doi parametri valorile întregi minimă și maximă pentru poziția glisorului. Opțional, puteți transmite valoarea `FALSE` ca un al treilea parametru pentru a preveni redesenarea automată a controlului. Există, de asemenea, funcții pentru stabilirea individuală a limitelor inferioară și superioară ale intervalului, în acest scop transmițându-se un întreg în apelul funcțiilor `SetRangeMin()` sau `SetRangeMax()`. Funcțiile reciproce `GetRangeMin()` și `GetRangeMax()` întorc limitele curente ale intervalului de valori.

Determinarea canalului rectangular al glisorului

Canalul rectangular reprezintă acea suprafață a controlului glisor de-a lungul căreia se deplasează cursorul. Dimensiunea și poziția acestei suprafețe rectangulare pot fi determinate apelând funcția `GetChannelRect()` a clasei `CSliderCtrl`.

Poziția cursorului poate fi stabilită prin apelul funcției `SetPos()`, transmițându-se o valoare întreagă care reprezintă noua poziție din cadrul intervalului, și poate fi determinată prin valoarea întoarsă de funcția membru `GetPos()`.

Ca și în cazul controlului bară de derulare, utilizatorul poate incrementa poziția glisorului cu valori de linie sau de pagină, apăsând tastele săgeată sau, respectiv, tastele `Page Up/Page Down`. Cu toate acestea, nu mai trebuie să tratați explicit mesajele trimise de controlul glisor pentru a actualiza poziția acestuia, așa cum era cazul barelor de derulare. Pentru a controla prin program valorile de increment pentru deplasarea cu o linie sau cu o pagină puteți apela funcțiile `SetLineSize()` și `SetPageSize()` oferite de către control, transmițând o valoare întreagă corespunzătoare pe post de dimensiune de linie sau de pagină. Valorile curente pentru dimensiunea de pagină și de linie sunt întoarse de funcțiile membru complementare `GetLineSize()` și `GetPageSize()`.

Este posibil, de asemenea, să stabiliți prin program frecvența de afișare a diviziunilor, în acest scop apelând funcția `SetTickFreq()` și transmițând frecvența dorită. Așadar, dacă doriți să afișați diviziuni din cinci în cinci unități, scrieți următoarea linie:

```
m_Slider.SetTickFreq(5);
```

Pentru ca funcția `SetTickFreq()` să aibă efect trebuie să fi selectat anterior stilul **Auto Ticks** în cadrul paginii **Style** din caseta de dialog **Slider Properties**. Puteți afla numărul de diviziuni existente în intervalul specificat pe baza valorii întoarse de apelul funcției membru `GetNumTicks()`.

Înlăturarea prin program a diviziunilor

Puteți înlătura prin program orice diviziuni trasate în cadrul controlului glisor apelând la funcția `ClearTicks()`. Pentru a redesea controlul imediat după înlăturarea diviziunilor, transmiteți valoarea `TRUE` în apelul acestei funcții.

În cazul în care ați selectat opțiunea **Enable Selection** din pagina **Style** a casetei de dialog **Slider Properties**, puteți folosi funcția membru `SetSelection()` pentru a afișa un interval de selecție în cadrul controlului glisor, această selecție apărând sub forma unei bare colorate pe un fundal alb, sub glisor (vedeți figura 7.5). În apelul `SetSelection()` puteți transmite valorile minimă și maximă care mărginesc intervalul selectat în cadrul intervalului de valori pentru glisor. Selecția curentă poate fi determinată prin intermediul funcției `GetSelection()`, transmițându-i acesteia două referințe la variabilele de tip întreg care vor conține limitele inferioară și superioară ale selecției, sau puteți anula selecția apelând funcția `ClearSel()` a controlului.

VEDEȚI...

► Pentru mai multe informații privind inițializarea casetelor de dialog, vedeți pagina 204.

Tratarea mesajelor trimise de controlul glisor

Controalele glisor comunică cu fereastra părinte prin intermediul aceluiași mesaj `WM_VSCROLL` și `WM_HSCROLL` pe care le utilizau și barele de derulare. Singura diferență este că nu mai trebuie să stabiliți explicit noua poziție a glisorului, aceasta fiind actualizată automat pentru dumneavoastră. Puteți folosi, totuși, aceste mesaje pentru a efectua operații specifice atunci când utilizatorul modifică poziția glisorului.

Spre exemplu, dacă doriți să reflectați poziția unui glisor în cadrul unui control indicator de evoluție, așa cum o arată figura 7.5, ați putea adăuga cele două controale ca în figura 7.6. Ați putea apoi să mapați glisorul (având identificatorul `IDC_SLIDER1`) pe o variabilă control de tip `CSliderCtrl` numită `m_Slider`, și controlul indicator de evoluție (având identificatorul `IDC_PROGRESS1`) pe o variabilă control de tip `CProgressCtrl` numită `m_Progress`. Mai departe, ați putea să actualizați indicatorul de evoluție de fiecare dată când poziția controlului glisor este modificată adăugând următoarea funcție de tratare `OnHScroll()`, împreună cu codul aferent:

```
void CSliderDlg::OnHScroll(UINT nSBCode, UINT nPos,
    CScrollBar* pScrollBar)
{
    if (pScrollBar->GetDlgCtrlID() == IDC_SLIDER1)
        m_Progress.SetPos(m_Slider.GetPos());
    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```


Funcția `GetDlgCtrlID()` este folosită pentru a identifica glisorul, după care poziția controlului indicator de evoluție este stabilită cu ajutorul funcției membru `SetPos()` pe baza poziției glisorului obținută în urma apelului funcției `GetPos()`.

Aflarea unui control pe baza identificatorului său din cadrul casetei de dialog

`GetDlgItem()` este o funcție utilă complementară funcției `GetDlgCtrlID()`, care întoarce un pointer de tip `CWnd` către controlul cărui îi corespunde identificatorul transmis ca prim parametru. Dacă în caseta de dialog curentă nu este găsit nici un control având identificatorul specificat, este întoarsă valoarea `NULL`. Dacă doriți să utilizați pointerul `CWnd` întors ca și control anume, va trebui să-l convertiți la clasa MFC corespunzătoare acelui control.

Dacă doriți să urmăriți anumite deplasări de ordinul liniilor sau al paginilor, puteți compara parametrul `nsbCode` cu valorile din tabelul 7.2, exact ca în cazul barelor de derulare.

VEDEȚI...

► Pentru informații privind identificatorii controalelor și mesaje de informare, vedeți pagina 75.

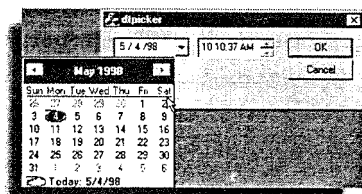
Utilizarea controalelor de selecție a datei/orei

Majoritatea utilizatorilor ar protesta dacă ar fi nevoiți să selecteze data sau ora prin intermediul unui glisor, dar până la Visual C++ 6.0 nu existau controale integrate pentru selecția datei și a orei; trebuia să implementați un mecanism propriu în acest scop sau să utilizați un control ActiveX. Acum Microsoft a adăugat în paleta Control două controale interconectate – selectorul de dată/oră și calendarul lunar. Controlul de selecție a datei/orei folosește automat controlul calendar (despre care vom discuta în secțiunea următoare) pentru a permite utilizatorului să aleagă o dată în urma efectuării unui clic pe un buton săgeată de derulare.

Se poate să fi întâlnit deja aceste controale la lucru în aplicația de poștă electronică Microsoft Outlook. Selectorul de dată/oră poate fi utilizat pentru alegerea unei date sau a unei ore (pentru a permite utilizatorului să specifice ambele repere puteți utiliza două controale). Atunci când este folosit pentru alegerea unei date, selectorul de dată/oră apare în mod normal sub forma unei casete combinate, afișând data selectată curent în format lung sau scurt. La efectuarea unui clic pe săgeata de derulare a casetei combinate, este afișat un control calendar (ilustrat în figura 7.7) care vă permite să precizați luna și ziua. Versiunea destinată selectării unei ore (înfățișată în dreapta figurii 7.7) afișează ora selectată curent și vă permite modificarea acesteia fie prin editare directă, fie prin intermediul controlului de modificare incrementală.

FIGURA 7.7

Selectorul de dată/oră utilizat pentru a permite specificarea unei date și a unei ore.



VEDEȚI...

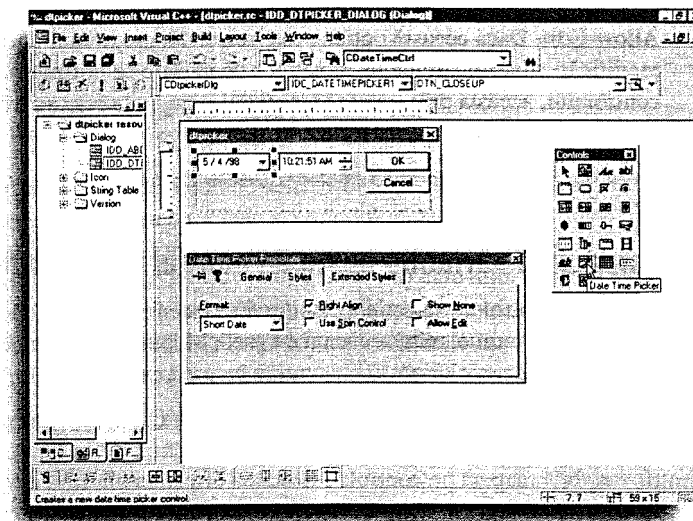
► Pentru un exemplu de control ActiveX de selecție a datei, vedeți pagina 187.

Adăugarea unui control de selecție a datei/orei într-o casetă de dialog

Controlul de selecție a datei/orei poate fi preluat din paleta Controls (fiind evidențiat în figura 7.8 de eticheta explicativă) și plasat pe macheta unei casete de dialog, fiind permisă dimensionarea și poziționarea sa corespunzătoare. Ca și în cazul celorlalte controale, este bine să fixați un identificator unic în cadrul paginii General a casetei de dialog Date Time Picker Properties (ce poate fi apelată selectând controlul și apăsând `Alt+Enter`), casetă ilustrată în figura 7.8.

FIGURA 7.8

Adăugarea controalelor de selecție a datei/orei într-o casetă de dialog în cadrul editorului de resurse.



Pagina **Styles** a casetei de dialog Date Time Picker Properties vă permite să stabiliți unul din modurile de selecție a datei sau a orei prin alegerea uneia dintre următoarele trei opțiuni prezente în lista derulantă **Format**:

- **Short Date.** Această opțiune determină controlul să afișeze data calendaristică sub o formă prescurtată (de pildă, 5/4/98), efectuarea unui clic pe butonul de derulare al controlului având ca efect afișarea unui calendar care vă permite alegerea unei date precise.
- **Long Date.** Această opțiune determină controlul să afișeze data calendaristică sub o formă detaliată (de exemplu, Monday, May 04, 1998), efectuarea unui clic pe butonul de derulare ducând și de data aceasta la afișarea controlului calendar.
- **Time.** Această opțiune determină afișarea orei (de pildă, 10:21:51 AM) și a unui control de modificare incrementală în locul butonului de derulare, control care permite ajustarea orei, a minutelor sau a secundelor din cadrul reperului orar.

Puteți stabili, de asemenea, următoarele opțiuni de stil efectuând clic pe casetele de validare corespunzătoare pentru a comuta starea opțiunii:

- **Right Align.** Dacă această opțiune este selectată, la afișarea controlului calendar în urma efectuării de către utilizator a unui clic pe butonul de derulare al controlului, acesta va fi aliniat la dreapta selectorului de dată/oră; altfel, calendarul va fi afișat dedesubtul controlului.
- **Use Spin Control.** Dacă această opțiune este selectată, în locul butonului de derulare va fi afișat un control de modificare incrementală. Utilizatorul poate efectua clic pe săgețile sus și jos ale acestui control pentru a modifica ziua, luna sau anul din cadrul datei.
- **Show None.** Dacă această opțiune este selectată, în stânga selectorului de dată/oră este afișată o casetă de validare care permite utilizatorului să precizeze că nu este selectată nici o dată și nici o oră atunci când caseta de validare a selectorului nu este bifată.
- **Allow Edit.** Dacă această opțiune este selectată, programul poate să efectueze diferite modificări și verificări ale informațiilor pe măsură ce utilizatorul le editează în cadrul controlului, aceasta prin tratarea mesajului `DTN_USERSTRING` trimis de către control.

VEDEȚI...

- Pentru mai multe detalii privind editarea casetelor de dialog și proprietățile controalelor, vedeți pagina 54.

Maparea unei variabile peste un control de selecție a datei/orei

Maparea unui control de selecție a datei/orei se face în maniera cunoscută prin intermediul `ClassWizard`, așa cum o arată secvența de pași „Maparea unei variabile membru peste un control indicator de evoluție“, mai devreme în acest capitol.

În cazul în care selectați **Value** în caseta combinată **Category** (din pasul 5), caseta combinată **Variable Type** vă va permite să alegeți fie `CTime`, fie `COleDateTime` ca tip pentru variabila mapată pe control. Selectarea oricăreia dintre aceste două clase similare pentru operarea cu repere temporale oferă o modalitate rapidă și simplă pentru fixarea sau determinarea unei valori înscrise în controlul de selecție a datei/orei, folosind variabila membru mapată (indiferent de tipul ei) așa cum ați utiliza o variabilă `CString` mapată pe un control de editare.

Utilizarea claselor `CTime` și `COleDateTime`

Este de preferat să utilizați clasa `COleDateTime` în locul clasei `CTime` deoarece permite reprezentarea datelor între 1 ianuarie 100 și 31 decembrie 9999. În schimb, `CTime` poate reține numai date între 1 ianuarie 1970 și 19 ianuarie 2038. De asemenea, așa cum o sugerează și numele său, `COleDateTime` poate fi utilizată mai ușor în interacțiunea cu programele OLE și COM.

Dacă aveți nevoie de un control mai bun prin program asupra selectorului de dată/oră, ar trebui să alegeți **Control** în caseta combinată **Category**, ceea ce va înscrise automat în câmpul **Variable Type** numele clasei `CDateTimeCtrl` (care încapsulează controlul de selecție a datei/orei, așa cum veți vedea pe parcursul acestei secțiuni).

Puteți destul de bine să mapați două sau mai multe variabile de tipuri diferite peste un același control (după identificator) astfel încât să beneficiați de avantajele ambelor situații: control bun prin clasa `CDateTimeCtrl` și acces rapid printr-un obiect de tip `COleDateTime`.

Odată selectat tipul variabilei pe care o veți mapa peste control, puteți efectua clic pe **OK** pentru a închide caseta de dialog `Add Member Variable` și `ClassWizard`. În consecință, noua variabilă (noile variabile) vor fi inserate în clasa corespunzătoare pe care ați specificat-o (de regulă, clasa care gestionează caseta de dialog).

În cazul aplicației de tip dialog generată cu `AppWizard`, pe care am folosit-o în această secțiune, puteți să plasați două controale (unul pentru dată și unul pentru oră), ca în figura 7.8. Controlul destinat selecției datei este referit prin identificatorul `IDC_MYDATA` și mapat pe variabila `m_myDate`, iar cel destinat selecției orei are identificatorul `IDC_MYTIME` și este mapat pe variabila `m_myTime`. Ambele variabile sunt instanțe ale clasei `CDateTimeCtrl`.

VEDEȚI...

- Pentru o explicație mai amănunțită privind maparea variabilelor membru, vedeți pagina 206.

Inițializarea unui control de selecție a datei/orei

Asemeni controalelor glisor și bară de derulare, selectorul de dată/oră permite stabilirea unui interval. Acest interval are forma unor date și ore minime și maxime permise. Fixarea intervalului se poate face prin apelul funcției `SetRange()` a clasei `CDateTimeCtrl`, transmițând doi pointeri la obiecte `COleDateTime` (sau `CTime`) care rețin limitele inferioară și superioară de dată-oră. Spre exemplu, dacă doriți să limitați un control de selecție a datei/orei pentru a permite utilizatorului să aleagă exclusiv repere temporale din anul 1998, la sfârșitul funcției `OnInitDialog()` ați putea adăuga următoarele linii de cod:

```
COleDateTime dtMin(1998,1,1,0,0,0);
COleDateTime dtMax(1998,12,31,23,59,59);
m_myDate.SetRange(&dtMin,&dtMax); // Fixam intervalul pentru control
```

Starea de validitate a obiectului `COleDateTime`

Dacă ați generat un obiect `COleDateTime` pe baza unui reper dată-oră corect, obiectul va fi marcat printr-un indicator de validitate (`COleDateTime::valid`). În caz contrar, indicatorul de validitate este inhibat (`COleDateTime::invalid`). Puteți testa acest indicator apelând funcția membru `GetStatus()`, care întoarce valoarea indicatorului, sau puteți să forțați o valoare pe care o transmiteți în apelul funcției `SetStatus()`.

Primele două linii construiesc obiectele `COleDateTime` cu parametri specificând anul, luna, ziua, ora, minutul și secunda. Apoi, prin intermediul funcției `SetRange()`, sunt transmiși pointeri la aceste două obiecte variabilei `m_myDate` (de tip `CDateTimeCtrl`), care a fost mapată pe control. La execuție, utilizatorul nu va putea să aleagă nici o dată în afara anului 1998, iar datele aflate în afara intervalului vor fi afișate voalat (la apelarea controlului calendar). O funcție `GetRange()` complementară înscrise în două obiecte `COleDateTime` (sau `CTime`) limitele intervalului definit.

Puteți, de asemenea, să specificați modul în care selectorul de dată/oră afișează data și ora selectate, modificând formatul existent. Pentru a modifica formatul implicit de afișare trebuie să transmiteți funcției `SetFormat()` un șir de caractere corespunzător, șir construit din codurile de formatare prezentate în tabelul 7.3.

TABELUL 7.3 Coduri de formatare pentru controlul de selecție a datei/orei

Cod de formatare	Descriere
YYY	Afișează anul complet (de exemplu, '1998')
YY	Afișează ultimele două cifre ale anului (de exemplu, '98')
Y	Afișează ultima cifră a anului (de exemplu, '8')
MMMM	Afișează numele complet al lunii (de exemplu, 'April')
MMM	Afișează prescurtarea din trei litere a lunii (de exemplu, 'Apr')
MM	Afișează numărul lunii pe două cifre (de exemplu, '04')
M	Afișează numărul lunii pe una sau două cifre (de exemplu, '4' pentru aprilie și '11' pentru noiembrie)
dddd	Afișează numele complet al zilei din săptămână (de exemplu, 'Monday')
ddd	Afișează prescurtarea din trei litere a zilei din săptămână (de exemplu, 'Mon')
dd	Afișează numărul zilei din lună pe două cifre (de exemplu, '04')
d	Afișează numărul zilei din lună pe una sau două cifre (de exemplu, '4' pentru a patra zi și '11' pentru a unsprezecea zi)
HH	Afișează ora pe două cifre, în formatul de 24 de ore (de exemplu, '16')
hh	Afișează ora pe două cifre, în formatul de 12 de ore (de exemplu, '04')
H	Afișează ora pe una sau două cifre, în formatul de 24 de ore (de exemplu, '4' pentru ora 4 AM și '16' pentru ora 4 PM)
h	Afișează ora pe una sau două cifre, în formatul de 12 de ore (de exemplu, '4' pentru ora 4 AM și '4' pentru ora 4 PM)
tt	Afișează indicatorul AM/PM pe două caractere (de exemplu, 'AM' sau 'PM')
t	Afișează indicatorul AM/PM pe un caracter (de exemplu, 'A' sau 'P')
mm	Afișează minutele pe două cifre (de exemplu, '07' sau '59')
m	Afișează minutele pe una sau două cifre (de exemplu, '7' sau '59')

Formatarea reperelor din obiectele `COleDateTime`

Puteți preciza coduri de formatare similare și pentru obiectele `COleDateTime` (obținute prin intermediul controalelor de selecție a datei/orei) folosind funcția `Format()` a clasei. Această funcție primește ca prim parametru un șir de descriere a formatului.

Există o serie întreagă de coduri de formatare care tratează elemente precum luna, anul, ziua, ora, minutul și secunda. De exemplu, `%c` va avea ca rezultat data și ora formate după specificul zonei, ca aici: 04/04/98 18:05:01. În mod similar, o formă mai detaliată se poate obține prin intermediul specificatorului `%#c`, rezultatul fiind

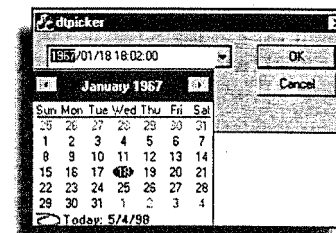
În șirul de descriere a formatului puteți adăuga propriile secvențe textuale, încadrându-le între caractere apostrof cu rol de delimitatori. Spre exemplu, dacă doriți să afișați data și ora în cadrul unui singur control de selecție, astfel încât utilizatorul să poată modifica fiecare componentă, și alegeți un format de genul 1998/05/04 12:22:23, ați putea adăuga în funcția `OnInitDialog()` următoarea linie de cod:

```
m_myDate.SetFormat("yyy'/'MM'/'dd' 'HH':'mm':'ss");
```

Observați că simbolurile slash, două puncte și spațiu sunt delimitate de caractere apostrof. La execuție, selectorul de dată/oră vă va permite să editați împreună data și ora, așa cum se vede în figura 7.9.

FIGURA 7.9

Un selector de dată/oră cu un format personalizat, afișând data și ora.



Ați putea dori, de asemenea, să stabiliți valorile inițiale ale orei și datei din cadrul selectorului, înlocuind inițializarea implicită care se face pe baza orei și datei curente din sistem. În acest scop puteți apela `SetTime()`, transmițând un obiect `COleDateTime` care conține data și ora cu care vreți să inițializați controlul.

De pildă, dacă vreți să stabiliți valorile inițiale la ora 6 PM și 2 minute, 18 ianuarie 1967, ați putea scrie următoarea linie de cod:

```
m_myDate.SetTime(COleDateTime(1967,1,18,18,2,0));
```

La afișarea controlului, veți vedea data și ora inițiale pe care le-ați stabilit, ca în figura 7.9.

Puteți, totodată, să determinați data și ora curente din cadrul selectorului, apelând funcția complementară `GetTime()` și transmițându-i un obiect `COleDateTime` care va fi completat de către control. Obiectul `COleDateTime` poate fi apoi folosit sau interogată după nevoie.

Spre exemplu, ați putea scrie următoarea secvență de cod care afișează o casetă de mesaj ce prezintă data și ora selectate în momentul în care se efectuează clic pe butonul **OK** al casetei de dialog (trebuie inserată în prealabil o funcție de tratare `OnOK()`):

```
void CDtpickerDlg::OnOK()
{
    ColeDateTime dtChosenTime;
    m_myDate.GetTime(dtChosenTime);
    AfxMessageBox(dtChosenTime.Format("%#x"));
    CDialog::OnOK();
}
```

Extragerea de valori dintr-un obiect COleDateTime

Există mai multe funcții de acces care întorc componente ale reperului temporar selectat, cum ar fi `GetYear()`, `GetMonth()`, `GetDay()`, `GetHour()`, `GetMinute()` sau `GetSecond()`. Există, în plus, o serie de funcții derivate care se pot dovedi utile: `GetDayOfWeek()` și `GetDayOfYear()`. `GetDayOfWeek()` întoarce o valoare între 1 și 7 care reprezintă poziția unei zile în cadrul săptămânii, 1 fiind corespunzător la duminică. Funcția `GetDayOfYear()` întoarce o valoare între 1 și 366, începând de la 1 ianuarie.

Data și ora curente sunt extrase și înscrise în variabila `dtChosenTime`, aceasta fiind apoi formatată prin intermediul funcției `Format()` a clasei `ColeDateTime` și afișată în cadrul unei casete de mesaj.

Selectorul de dată/oră vă permite, de asemenea, să stabiliți culoarea și fontul folosite la afișarea controlului calendar. Pentru a stabili culoarea unui element component puteți apela funcția `SetMonthCalColor()`, transmitând indexul elementului (preluat din tabelul 7.4 cu indecși) și o valoare de culoare. Transmiterea unui indicator valid către un font (HFONT) în apelul funcției `SetMonthCalFont()` duce la modificarea fontului utilizat implicit la afișarea calendarului.

VEDEȚI...

► Pentru informații referitoare la utilizarea fonturilor, vedeți pagina 393.

Tratarea mesajelor generate de modificarea datei

Atunci când utilizatorul modifică data înscrisă într-un control de selecție a datei/orei, puteți intercepta mesajul `DTN_DATETIMECHANGE` trimis casetei de dialog părinte pentru a efectua o operație specifică. Adăugarea unei funcții de tratare pentru acest mesaj se efectuează așa cum este descris în continuare.

Adăugarea unei funcții de tratare a mesajelor generate de modificarea datei

1. Efectuați un clic cu butonul drept pe controlul de selecție a datei/orei la care vreți să răspundeți în cadrul machetei casetei de dialog din editorul de resurse.
2. Selectați **Events** din meniul contextual afișat pentru a deschide caseta de dialog **New Windows Message and Event Handlers**.
3. Selectați mesajul `DTN_DATETIMECHANGE` din lista **New Windows Messages/Events**. (În alternativă, puteți selecta eventual unul dintre celelalte mesaje afișate.)

4. Efectuați un clic pe butonul **Add and Edit** pentru a afișa caseta de dialog **Add Member Function** conținând numele implicit al funcției de tratare.
5. Modificați numele funcției de tratare dacă găsiți de cuviință, după care efectuați un clic pe **OK** pentru a trece la editarea noii funcții de tratare, care va trebui să arate similară în conținut cu funcția prezentată în listingul 7.2.

Odată adăugată noua funcție de tratare, puteți efectua orice operație necesară în cadrul aplicației la modificarea datei sau a orei selectate. Funcția de tratare primește ca prim parametru un pointer la o structură `NMHDR`. În mod normal ați adăuga câte o funcție de tratare pentru fiecare selector de dată/oră; puteți, totuși, să verificați identificatorul controlului care a generat mesajul prin intermediul variabilei membru `idFrom` a structurii `NMHDR`. În acest fel puteți avea o singură funcție de tratare care răspunde la mai multe controale de selecție a datei/orei. Listingul 7.2 ilustrează o astfel de funcție de tratare partajată și cele două intrări `ON_NOTIFY` corespunzătoare din harta de mesaje, care referă aceeași funcție. Codul din listing modifică bara de titlu a casetei de dialog părinte, afișând aici data sau ora curentă de fiecare dată când utilizatorul alterează conținutul controalelor de selecție a datei și a orei.

LISTINGUL 7.2 LST7_2.CPP – Tratarea în mod partajat a mesajului `DTN_DATETIMECHANGE` generat de două controale de selecție a datei/orei

```
1 BEGIN_MESSAGE_MAP(CDtpickerDlg, CDialog)
2     //{AFX_MSG_MAP(CDtpickerDlg)
3     ON_WM_SYSCOMMAND()
4     ON_WM_PAINT()
5     ON_WM_QUERYDRAGICON()
6     ON_NOTIFY(DTN_DATETIMECHANGE, IDC_MYDATE,
7         OnDatetimechangeMydate)
8     //{AFX_MSG_MAP
9     ON_NOTIFY(DTN_DATETIMECHANGE, IDC_MYTIME,
10         OnDatetimechangeMydate)
11 END_MESSAGE_MAP()
12
13 void CDtpickerDlg::OnDatetimechangeMydate(
14     NMHDR* pNMHDR, LRESULT* pResult)
15 {
16     ColeDateTime dtSel;
17     switch(pNMHDR->idFrom)
18     {
19         case IDC_MYDATE:
20             m_myDate.GetTime(dtSel);
21             SetWindowText("Date:" + dtSel.Format("%#x"));
22             break;
23         case IDC_MYTIME:
24             m_myTime.GetTime(dtSel);
```

(continuare)

LISTINGUL 7.2 Continuare

```

22      SetWindowText("Time:" + dtSel.Format("%H:%M:%S"));
23      break;
24  }
25  *pResult = 0;
26  }

```

- ❶ A doua intrare din harta de mesaje este adăugată manual.
- ❷ În funcție de controlul care a generat mesajul, pentru actualizarea barei de titlu a casetei de dialog este folosit fie selectorul de dată (linia 16), fie selectorul de oră (linia 20).

Intrarea în harta de mesaje din linia 6 a listingului 7.2 este intrarea standard generată automat de ClassWizard (după parcurgerea secvenței de pași „Adăugarea unei funcții de tratare a mesajelor generate de modificarea datei”). În schimb, intrarea din linia 8 trebuie adăugată manual pentru a realiza partajarea aceleiași funcții de tratare (`OnDatetimechangeMydate()`).

Adăugarea manuală a intrărilor în harta de mesaje

De fiecare dată când adăugați manual o intrare în harta de mesaje, aveți grijă ca aceasta să fie plasată în exteriorul secțiunii încadrate de comentariile `// {AFX_MSG_MAP`. ClassWizard folosește aceste comentarii pentru a marca locul în care sunt inserate intrările generate automat și va fi derutat dacă vede că liniile de cod existente nu sunt exact cele la care se aștepta. Locul cel mai bun pentru adăugarea propriilor intrări în harta de mesaje este după cel de-al doilea comentariu `// {AFX_MSG_MAP`, dar înainte de macrodefiniția `END_MESSAGE_MAP()`.

Funcția de tratare `OnDatetimechangeMydate()` este implementată între liniile 11 și 26. În linia 13 este declarată o variabilă de tip `COleDateTime` (`dtSel`) care va reține data/ora selectată curent. Instrucțiunea `switch` din linia 14 are rolul de a determina controlul care a generat mesajul.

În cazul în care mesajul a fost trimis de controlul `IDC_MYDATE`, conținutul acestuia este extras în linia 17 prin intermediul variabilei mapate `m_myDate` și afișat în linia 18 pe bara de titlu a casetei de dialog, sub formă de dată.

Dacă expeditorul mesajului este controlul `IDC_MYTIME`, conținutul acestuia este extras în linia 21 prin intermediul variabilei mapate `m_myTime` și afișat în linia 22 pe bara de titlu a casetei de dialog, sub formă de oră.

VEDEȚI...

➤ Pentru informații privind identificatorii controalelor și mesajele generate, vedeți pagina 75.

Utilizarea controlului calendar

Controlul calendar poate fi folosit independent într-o casetă de dialog, ca orice alt control. Ați văzut acest calendar mai devreme, afișat la efectuarea unui clic pe butonul de derulare al selectorului de dată/oră. Este un control care generează o reprezentare asemănătoare

unui calendar, afișând zilele din fiecare lună. Utilizatorul poate selecta o anumită dată sau poate să modifice luna și anul afișate. În mod normal, controlul calendar permite selectarea unei singure date, însă poate fi configurat pentru a permite utilizatorului să specifice un interval de timp.

Adăugarea unui control calendar într-o casetă de dialog

Opțiunile de stil pentru controlul calendar

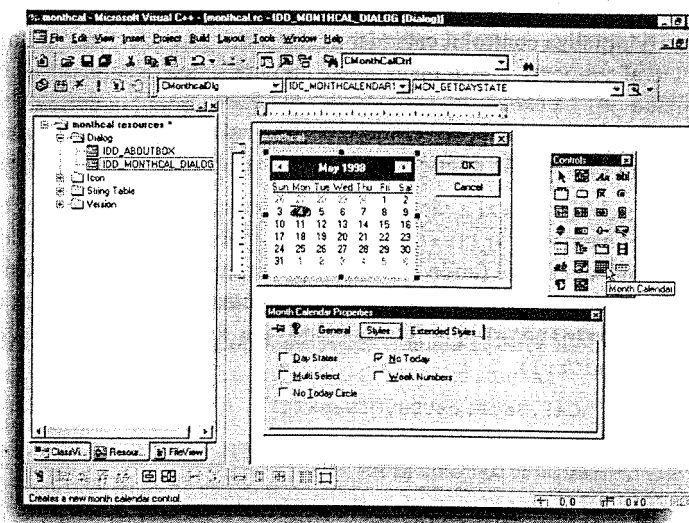
Aceste opțiuni de stil au echivalent în valorile simbolice `MCS_DAYSTATE`, `MCS_MULTISELECT`, `MCS_NOTODAY`, `MCS_NOTODAYCIRCLE` și `MCS_WEEKNUMBERS`. Ați putea dori să utilizați acești indicatori de stil în cazul în care creați un control dinamic, prin intermediul uneia dintre funcțiile sale `Create()`, și nu prin metoda obișnuită a machetelor casetelor de dialog.

Pictograma controlului calendar din cadrul paletelor Controls se află în dreapta celei corespunzătoare selectorului de dată/oră și poate fi selectată și plasată în maniera cunoscută. Puteți apela dialogul Month Calendar Properties selectând un astfel de control și apăsând `Alt+Enter`, ceea ce vă dă posibilitatea de a modifica identificatorul implicit al controlului prin intermediul paginii **General**. Pagina **Styles** (ilustrată în figura 7.10) vă permite stabilirea următoarelor opțiuni prin efectuarea de clic pe casetele de validare corespunzătoare:

- **DayStates**. Dacă această opțiune este selectată, puteți specifica prin program ce dată va fi afișată cu culori inverse (alb pe negru); altfel, data afișată invers va fi în mod automat data curentă din sistem.

FIGURA 7.10

Adăugarea unui control calendar în macheta unei casete de dialog prin intermediul editorului de resurse.



- **Multi Select.** Dacă această opțiune este selectată, utilizatorul va avea posibilitatea să specifice un interval de timp; în caz contrar, comportamentul implicit permite selecția unei singure date.
- **No Today.** Dacă această opțiune este selectată, linia roșie desenată în mod normal în jurul datei curente din sistem nu mai apare.
- **Week Numbers.** Dacă această opțiune este selectată, în stânga fiecăreia dintre săptămânile afișate va apărea și numărul acesteia în cadrul anului (între 1 și 52).

VEDEȚI...

- Pentru mai multe amănunte privind editarea casetelor de dialog și proprietățile controalelor, vedeți pagina 54.

Maparea unei variabile peste un control calendar

Operația de mapare este foarte similară cu maparea peste un control de selecție a datei/orei, despre care am discutat în secțiunea precedentă. Puteți mapa direct pe control o variabilă `CTime` sau `COleDateTime`, selectând opțiunea **Value** în caseta combinată **Category** din caseta de dialog **Add Member Variable**. Dacă aveți nevoie de un control mai bun, selectați opțiunea **Control** din caseta combinată **Category**, ceea ce va duce la selectarea automată a clasei `CMonthCalCtrl` în câmpul **Variable Type**, acesta fiind tipul variabilei mapate, variabilă membru al clasei casetei de dialog.

VEDEȚI...

- Pentru o explicație mai detaliată privind maparea variabilelor membru, vedeți pagina 206.

Inițializarea unui control calendar

Puteți inițializa controlul calendar pentru a mărgini selectarea în limitele unui anumit interval temporal, transmițând o dată minimă și una maximă în apelul funcției membru `SetRange()` a clasei `CMonthCalCtrl` sub forma unor pointeri la obiecte `COleDateTime` sau `CTime` (ca în cazul selectorului de dată/oră). Există o funcție reciprocă `GetRange()` care înscrie capetele intervalului în obiectele indicate de către doi parametri pointeri.

Puteți stabili prima zi a săptămânii, considerată implicit duminica, transmițând în apelul funcției membru `SetFirstDayOfWeek()` un indice care pornește de la zero și reprezintă numărul zilei dorite (unde luni este 0 și duminică este 6). Spre exemplu, dacă doriți ca luni să fie afișată ca prima zi a săptămânii, ați putea adăuga următoarea linie de cod la sfârșitul funcției `OnInitDialog()` (considerând că `m_MyMonthCal` este variabila mapată de tip `CMonthCalCtrl`):

```
m_MyMonthCal.SetFirstDayOfWeek(0);
```

Stabilirea pasului de derulare a lunilor

La derularea de către utilizator a controlului calendar, acesta parcurge de regulă lunile una câte una. Puteți modifica acest comportament implicit prin intermediul funcției `SetMonthDelta()`, transmițând numărul de luni cu care vreți să se producă derularea controlului. Funcția complementară `GetMonthDelta()` furnizează pasul de derulare curent al controlului.

Aveți posibilitatea de a modifica ziua de „astăzi” (încercuită cu roșu) transmițând funcției `SetToday()` un obiect `COleDateTime` (care să conțină nouă dată a zilei de astăzi). Data curentă a zilei de azi poate fi obținută transmițând o variabilă `COleDateTime` funcției complementare `GetToday()`. Această dată va fi înscrisă în respectiva variabilă.

Culorile implicite pot fi schimbate prin apelul funcției `SetColor()`, transmițând un index simbolic ca prim parametru (preluat din tabelul 7.4) și o valoare de culoare ca al doilea parametru.

TABELUL 7.4 Valorile simbolice pentru componentele de culoare ale controlului calendar

Valoarea simbolică	Descriere
<code>MCSC_TEXT</code>	Stabilește culoarea pentru afișarea datelor
<code>MCSC_TITLETEXT</code>	Stabilește culoarea pentru titlu
<code>MCSC_TITLENBK</code>	Stabilește culoarea pentru fundalul titlului
<code>MCSC_TRAILINGTEXT</code>	Stabilește culoarea pentru antet și pentru zilele externe
<code>MCSC_BACKGROUND</code>	Stabilește culoarea pentru fundal

VEDEȚI...

- Pentru detalii privind valorile de culoare, vedeți pagina 350.

Selecția unui interval de timp în cadrul controlului calendar

Puteți permite utilizatorului să selecteze o singură dată sau o întreagă perioadă de timp, în funcție de starea casetei de validare **Multi Select** din pagina **Styles** a casetei de dialog **Month Calendar Control Properties**.

Stabilirea intervalului de lucru pentru controlul calendar

Datele minimă și maximă permise în cadrul unui control calendar pot fi stabilite prin apelul funcției `SetRange()`, transmițând doi pointeri la obiecte de tip `COleDateTime`, așa cum o permitea și clasa `CDateTimeCtrl`. Funcția complementară `GetRange()` poate fi utilizată pentru a determina intervalul de lucru curent.

Atunci când se permite alegerea unei singure date, data selectată inițial poate fi transmisă funcției `SetCurSel()` a controlului calendar sub forma unui obiect `COleDateTime`, ca aici:

```
m_MyMonthCal.SetCurSel(COleDateTime(1967,1,18,18,2,0));
```

Data selectată sau modificată de către utilizator poate fi determinată în orice moment al existenței controlului transmițând un obiect `COleDateTime` în apelul funcției `GetCurSel()` a calendarului, ca aici:

```
COleDateTime dtChosenTime;
m_myData.GetCurSel(dtChosenTime);
AfxMessageBox(dtChosenTime.Format("%#x"));
```

Ultima linie va afișa data selectată într-o casetă de mesaj.

În cazul în care se permite selecția multiplă, va trebui să utilizați `GetSelRange()` și `SetSelRange()` pentru a determina intervalul selectat și, respectiv, pentru a stabili un nou interval selectat. Ambele funcții primesc ca parametri două obiecte `COleDateTime`. Primul parametru reprezintă limita inferioară a intervalului selectat, iar cel de-al doilea reprezintă limita superioară a acestui interval.

Selectarea unui interval este ceva mai complicată prin aceea că, implicit, selecția nu poate cuprinde mai mult de șapte zile. Puteți înlocui, însă, această valoare implicită cu orice altă limită a numărului de zile selectate, transmițând noua valoare întreagă în apelul funcției `SetMaxSelCount()` a calendarului.

Spre exemplu, dacă ați dori să inițializați controlul calendar pentru a afișa ca selectate primele 14 zile din ianuarie 1998, permițând utilizatorului să selecteze cel mult 15 zile consecutive, ați putea adăuga următoarele linii de cod:

```
m_MyMonthCal.SetMaxSelCount(15);
m_MyMonthCal.SetSelRange(COleDateTime(1998,1,1,0,0,0),
                           COleDateTime(1998,1,14,0,0,0));
```

La execuție, calendarul va afișa perioada selectată și vă va permite să selectați un interval de cel mult 15 zile.

Ați putea determina apoi ce interval a fost selectat de utilizator și câte zile cuprinde acesta adăugând liniile următoare:

```
COleDateTime dtMin, dtMax;
m_MyMonthCal.GetSelRange(dtMin,dtMax);
COleDateTimeSpan dtSpan = dtMax - dtMin;
AfxMessageBox(dtSpan.Format("You Selected %D Days"));
```

Numărul de zile este calculat scăzând din limita superioară a selecției data ce reprezintă limita inferioară, iar această diferență este înscrisă în obiectul `dtSpan` având tipul `COleDateTimeSpan`. După aceea este afișată o casetă de mesaj care conține numărul zilelor selectate.

Tratarea mesajelor generate de modificarea datei selectate

Puteți adăuga o funcție care să fie apelată în momentul în care utilizatorul modifică data selectată. Pașii sunt cei din secvența „Adăugarea unei funcții de tratare a mesajelor generate de modificarea datei”, cu deosebirea că mesajul interceptat este `MCN_SELCHANGE`. La adăugarea unei funcții de tratare a acestui mesaj, corpul funcției va arăta astfel:

```
void CMonthCalDlg::OnSelchangeMonthCalendar1(
    NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: ...
    *pResult = 0;
}
```

Utilizarea obiectelor `COleDateTimeSpan`

Puteți stabili un interval de timp transmițând numărul de zile, ore, minute și secunde ca parametri întregi în apelul funcției `SetDateTimeSpan()`. Puteți, de asemenea, să extrageți aceste valori dintr-un obiect `COleDateTimeSpan` valid apelând funcțiile `GetDays()`, `GetHours()`, `GetMinutes()` și `GetSeconds()`, care întorc întregi lungi reprezentând componenta respectivă a intervalului de timp. Dacă doriți să determinați întreaga perioadă de timp exprimată în zile, ore, minute sau secunde, sub forma unei valori double, puteți apela respectiv la funcțiile `GetTotalDays()`, `GetTotalHours()`, `GetTotalMinutes()` și `GetTotalSeconds()`.

Puteți adăuga în continuare propriul cod, inspectând structura `NMHDR` pentru a determina controlul care a generat mesajul, așa cum se arată în listingul 7.2. Puteți, de asemenea, să scrieți cod pentru a determina selecția curentă, folosind funcțiile `GetCurSel()` și `GetSelRange()` prezentate în secțiunea anterioară.

VEDEȚI...

- Pentru informații suplimentare privind identificatorii controalelor și mesajele de informare, vedeți pagina 75.

Tratarea evenimentelor generate de mouse

Scrierea codului pentru tratarea evenimentelor de apăsare și eliberare a butoanelor mouse-ului de către utilizator

Tratarea deplasării mouse-ului prin intermediul propriilor funcții

Folosirea coordonatelor mouse-ului și verificarea încadrării într-o zonă selectată

Urmărirea butoanelor mouse-ului

De multe ori v-ați putea dori să știți când a fost apăsat sau eliberat un buton al mouse-ului și unde a fost apăsat – fără a utiliza un control de tip buton. Un exemplu evident este o aplicație de desenare în care o parte din fereastră este utilizată ca suprafață de lucru și clicurile de mouse sunt folosite pentru a permite desenarea în interiorul acestei suprafețe. Așa cum ați văzut deja, toate acțiunile utilizatorului ajung la aplicație sub formă de mesaje (sau evenimente). Mesajele generate de acționarea butoanelor mouse-ului se conformează și ele regulii generale. În aceste mesaje sunt împachetate informații referitoare la poziția mouse-ului, ceea ce le face foarte ușor de utilizat în conjuncție cu funcțiile de desenare din Windows.

Urmărirea momentelor în care sunt apăstate butoane sau taste

Toate mesajele Windows sunt însoțite de un reper temporal care indică momentul în care mesajul a fost generat. Această informație poate fi obținută prin intermediul membrului `time` al structurii `MSG`, fiind necesară supradefinirea funcției virtuale `PreTranslateMessage()`. `PreTranslateMessage()` este apelată înaintea prelucrării pe baza hărții de mesaje a mesajelor primite de aplicație.

Tratarea mesajelor generate de apăsarea și eliberarea butoanelor mouse-ului

Windows urmărește trei butoane ale mouse-ului: butonul stâng, cel drept și cel din mijloc. Unele mouse-uri dispun de numai două butoane, caz în care vor exista numai evenimentele corespunzătoare butonului stâng și drept. Din această cauză, majoritatea aplicațiilor răspund la butoanele stâng și drept, dar pot exista situații speciale în care să doriți utilizarea tuturor celor trei butoane. Pentru fiecare buton, un program primește de la Windows două evenimente: unul corespunzător apăsării butonului de către utilizator și celălalt corespunzător eliberării butonului. Tratarea acestor evenimente se mai numește și *interceptare*. Pentru că aceste mesaje sunt trimise în mod normal clasei de bază a ferestrei, se spune că le interceptați sau le prindeți prin propriul cod.

Pentru a experimenta interceptarea acestor evenimente, creați o aplicație simplă de tip dialog cu ajutorul AppWizard și denumiți-o `MouseMsg`. (Urmați secvența de pași „Crearea unui nou proiect pentru o aplicație de tip dialog” din cadrul capitolului 3, „Crearea și proiectarea casetelor de dialog“.)

Odată creată aplicația de tip dialog, ar fi bine să măriți dimensiunea dialogului principal al aplicației, `IDD_MOUSEMSG_DIALOG`, la aproximativ 300 de pixeli pe lățime și 200 de pixeli pe înălțime, folosind în acest scop editorul de resurse. Astfel veți avea suficient loc pentru a inspecta coordonatele determinate ale mouse-ului. Dacă tot vă ocupați de editarea casetei de dialog, înlăturați și eticheta `TODO: Insert Controls` astfel încât să rămâneți cu o suprafață goală întinsă, pe lângă care se mai află doar butoanele OK și Cancel.

Prin intermediul secvenței de pași de mai jos puteți adăuga o funcție de tratare a mesajelor care să intercepteze mesajul generat la apăsarea de către utilizator a butonului stâng al mouse-ului.

Adăugarea unei funcții de tratare a mesajului generat de apăsarea butonului mouse-ului

1. Efectuați un clic pe eticheta paginii **ClassView** a secțiunii spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul + aflat în partea superioară pentru a afișa clasele din proiect.
3. Efectuați un clic cu butonul drept pe clasa **CMouseMsgDlg** pentru a apela meniul de context.
4. Selectați opțiunea **Add Windows Message Handler** pentru a deschide caseta de dialog **New Windows Message and Event Handlers**.
5. Selectați mesajul **WM_LBUTTONDOWN** din lista **New Windows Messages/Events**.
6. Efectuați un clic pe butonul **Add and Edit** pentru a edita noua funcție de tratare.

Acum aveți o funcție de tratare a mesajelor care se numește **OnLButtonDown()** și răspunde la apăsarea butonului stâng al mouse-ului. Această funcție va fi apelată de fiecare dată când utilizatorul apasă butonul stâng al mouse-ului. Puteți modifica această funcție pentru a insera cod propriu care să deseneze ceva. De pildă, dacă adăugați codul prezentat în listingul 8.1, va fi desenată pictograma unui semn de exclamare în fiecare loc în care este apăsat butonul stâng al mouse-ului.

LISTINGUL 8.1 LST8_1.CPP – Personalizarea funcției de tratare OnLButtonDown() în sensul desenării unui semn de exclamare în poziția în care s-a efectuat clic

```

1 void CMouseMsgDlg::OnLButtonDown(UINT nFlags,
  CPoint point)
2 {
3     // TODO: Add your message handler code here and /or
4
5     // ** Afisam un mesaj pe bara de titlu a ferestrei
6     CString strMessage;
7     strMessage.Format("Left Button Pressed at (%d,%d)",
8                       point.x, point.y);
9     SetWindowText(strMessage);
10
11    // ** Selectam o pictograma in functie de tasta Ctrl
12    char* pszIcon;
13    if (nFlag & MK_CONTROL)
14        pszIcon = IDI_EXCLAMATION;
15    else
16        pszIcon = IDI_HAND;
17
18    // ** Desenam pictograma selectata in punctul apasarii
19    CDC* pDC = GetDC();

```

```

20    pDC->DrawIcon(point,
21                  AfxGetApp()->LoadStandardIcon(pszIcon));
22    ReleaseDC(pDC);
23
24    CDialog::OnLButtonDown(nFlags, point);
25 }

```

- 1 Poziția este înscrisă într-o variabilă șir de caractere și afișată pe bara de titlu a dialogului.
- 2 Dacă a fost apăsată tasta Ctrl, este selectată o altă pictogramă.
- 3 Pictograma selectată este afișată în locul în care s-a efectuat clic.

Observați în listingul 8.1 că funcția de tratare **OnLButtonDown()** primește doi parametri: primul parametru, **nFlags**, arată ce alte butoane ale mouse-ului mai sunt apăstate și dacă au fost apăstate tastele Ctrl și Shift, după cum este descris în tabelul 8.1. Aceste valori sunt *măști de biți* și la un moment dat poate fi întâlnită orice combinație. Pentru a verifica un anumit indicator puteți utiliza operatorul **&** logic (operatorul **&** din C++), așa cum se vede în linia 13. Aici este verificat indicatorul **MK_CONTROL**, care va fi **TRUE** în cazul în care la efectuarea clicului a fost apăsată și tasta Ctrl.

În linia 14 din listing, pointerului **pszIcon** îi este atribuită una dintre pictogramele **IDI_EXCLAMATION** sau **IDI_HAND**, în funcție de rezultatul testului efectuat asupra tastei Ctrl prin intermediul parametrului **nFlags**. Aceste pictograme sunt pictograme standard ale sistemului și sunt utilizate în liniile 20 și 21 în cadrul funcției **DrawIcon()** a contextului dispozitiv. (Nu aș vrea să intru acum în detalii privind contextele dispozitiv; vom vorbi despre acestea în capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv“.) În linia 21, funcția **LoadStandardIcon()** a clasei aplicației primește pointerul **pszIcon** pentru a încărca pictograma standard corespunzătoare; alte pictograme standard sunt prezentate în tabelul 8.2.

Cel de-al doilea parametru transmis funcției de tratare **OnLButtonDown()** este un obiect **CPoint**, numit **point**. **CPoint** este o clasă care conține coordonatele unui punct sub forma membrilor **x** și **y**. Linia 7 construiește pe baza acestui punct un șir care va fi afișat pe bara de titlu a casetei de dialog cu ajutorul apelului **SetWindowText()** din linia 9. Același punct este folosit în linia 20 în scopul plasării pictogramei.

TABELUL 8.1 Indicatori stabiliți în cadrul parametrului nFlags al funcției OnLButtonDown

Valoarea indicatorului	Descriere
MK_LBUTTON	Arată că este apăsat butonul stâng al mouse-ului
MK_RBUTTON	Arată că este apăsat butonul drept al mouse-ului
MK_MBUTTON	Arată că este apăsat butonul din mijloc al mouse-ului
MK_CONTROL	Arată că este apăsată tasta Ctrl
MK_SHIFT	Arată că este apăsată tasta Shift

Pictogramele standard din Windows 95 și NT 4.0

Dacă utilizați aceste pictograme standard, veți vedea că aspectul nu corespunde numelui. Spre exemplu, `IDI_HAND` corespunde unui cruce albe pe fond roșu, fără nici o urmă de mână (hand). Cauza provine de la Windows 3.11 și NT 3.51; aspectul pictogramelor a fost actualizat, dar numele au rămas aceleași și nu se mai potrivesc cu aspectul.

TABELUL 8.2 Pictogramele standard ale sistemului, accesibile prin intermediul funcției `LoadStandardIcon()`

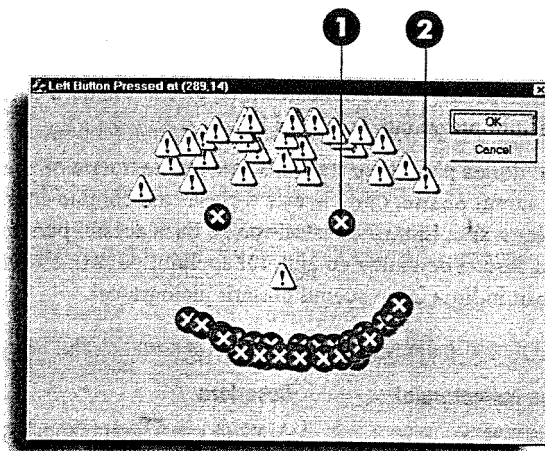
Pictogramă	Descriere
<code>IDI_EXCLAMATION</code>	Un semn de exclamare în interiorul unui triunghi
<code>IDI_HAND</code>	Un semn X într-un cerc roșu.
<code>IDI_APPLICATION</code>	Pictograma implicită a aplicației
<code>IDI_ASTERISK</code>	Un i mic, de la informații
<code>IDI_QUESTION</code>	Un semn de întrebare

Dacă inserați aceste linii de cod în funcția de tratare, așa cum o ilustrează listingul 8.1, după asamblarea și rularea aplicației veți avea cel mai simplu program de desenare posibil. Ar trebui ca la efectuarea unui clic pe suprafața casetei de dialog să fie desenat fie un semn de exclamare, dacă ați apăsat doar butonul stâng al mouse-ului, fie o cruce, dacă ați apăsat butonul stâng al mouse-ului împreună cu tasta Ctrl, așa cum se vede în figura 8.1.

FIGURA 8.1

Pictogramele desenate la apăsarea butonului stâng al mouse-ului.

- 1 Pictograma cu cruce este desenată atunci când a fost apăsată tasta Ctrl.
- 2 Semnul de exclamare este desenat atunci când se apasă doar butonul stâng al mouse-ului.



Observați, de asemenea, că poziția curentă a mouse-ului este afișată pe bara de titlu ca fiind poziția în care s-a efectuat ultimul clic. Dacă efectuați un clic lângă colțul din stânga sus, veți vedea că valorile coordonatelor sunt relative la colțul stânga sus al ferestrei dialog, nefiind coordonate absolute în cadrul ecranului. Coordonatele absolute se pot determina folosind în loc funcția `GetCursorPos()`, care poate fi apelată în orice moment

pentru a obține poziția curentă a cursorului mouse-ului. Această funcție primește un singur parametru, un pointer la obiectul `CPoint` în care va fi înscrisă poziția cursorului. Puteți testa această funcție adăugând un apel `GetCursorPos()` în interiorul funcției de tratare `OnLButtonDown()`, după comentariul `// TODO: ...` din linia 3 a listingului 8.1, ca aici:

```
GetCursorPos(&point);
```

Dacă asamblați și rulați programul după adăugarea acestei linii, veți vedea că perechea de coordonate afișată pe bara de titlu este relativă acum la colțul stânga sus al ecranului, nu la cel al casetei de dialog. De asemenea, pictogramele vor fi desenate în funcție de acest deplasament. Cum funcția de desenare se așteaptă la poziții relative la fereastră, pictogramele vor apărea la o oarecare distanță de cursorul mouse-ului. Pentru a reveni la poziționarea corectă a pictogramelor va trebui să eliminați linia adăugată după ce i-ați observat efectul.

Puteți crea o funcție de tratare care să intercepteze mesajul generat de eliberarea butonului stâng al mouse-ului urmând secvența de pași anterioară care a adăugat funcția de răspuns la apăsarea butonului, cu mențiunea că în locul mesajului `WM_LBUTTONDOWN` va trebui să selectați mesajul `WM_LBUTTONUP`. Rezultatul va fi crearea unei funcții `OnLButtonUp()`. Ați putea adăuga apoi propriul cod care să afișeze pe bara de titlu un mesaj ce indică eliberarea butonului stâng al mouse-ului, ca aici:

```
SetWindowText("Left Button Released");
```

Dacă adăugați această linie și apoi asamblați și rulați programul, veți vedea că noua funcție de tratare este apelată atunci când eliberați butonul stâng al mouse-ului. Aveți în acest fel posibilitatea de a scrie cod care să aducă aplicația într-o stare specială atunci când apăsați butonul, revenind la normal odată cu eliberarea butonului. Un exemplu de zi cu zi al unui astfel de comportament este deplasarea ferestrelor pe suprafața de lucru. Apăsarea butonului stâng în timp ce cursorul se află deasupra barei de titlu a unei ferestre declanșează starea de deplasare; eliberarea butonului duce la fixarea poziției ferestrei și la reluarea funcționării normale.

Există funcții de tratare similare pentru evenimentele generate de acționarea butoanelor drept și din mijloc. Tabelul 8.3 prezintă toate evenimentele legate de butoanele mouse-ului. Crearea unor funcții care să răspundă la butoanele drept și din mijloc se face prin intermediul aceluiași pași pe care i-ați folosit deja pentru a trata evenimentele ce privesc butonul stâng al mouse-ului.

Utilizarea funcțiilor de tratare a butonului din mijloc

Adăugarea unei funcții de tratare pentru butonul din mijloc este mai delicată, deoarece Microsoft nu a prevăzut asistentul specializat cu posibilitatea de a genera o astfel de funcție. Aveți oricum posibilitatea de a crea propriile funcții de tratare pentru aceste mesaje, dar veți fi nevoit să adăugați manual intrarea din harta de mesaje și definiția funcției. Scopul urmărit este de a descuraja dezvoltatorii în a crea programe care funcționează numai cu mouse-uri cu trei butoane.

TABELUL 8.3 Mesajele generate în Windows la acționarea butoanelor mouse-ului

Numele mesajului	Numele funcției de tratare	Descriere
WM_LBUTTONDOWN	OnLButtonDown()	Apăsarea butonului stâng
WM_LBUTTONUP	OnLButtonUp()	Eliberarea butonului stâng
WM_MBUTTONDOWN	OnMButtonDown()	Apăsarea butonului din mijloc
WM_MBUTTONUP	OnMButtonUp()	Eliberarea butonului din mijloc
WM_RBUTTONDOWN	OnRButtonDown()	Apăsarea butonului drept
WM_RBUTTONUP	OnRButtonUp()	Eliberarea butonului drept

VEDEȚI...

- Pentru mai multe informații privind desenarea în cadrul contextelor dispozitiv, vedeți pagina 323.
- Pentru a afla cum se creează și se editează machetele casetelor de dialog, vedeți pagina 54.

Interceptarea evenimentelor dublu clic

Dublul clic are de multe ori o semnificație specială în cadrul aplicațiilor Windows. Este folosit adesea ca o operație scurtătură în efectuarea de selecții din liste. Utilizatorul poate trimite un eveniment dublu clic către o aplicație printr-o succesiune rapidă de două clicuri ale unui buton al mouse-ului. Windows testează intervalul de timp dintre cele două clicuri și generează un mesaj `WM_LBUTTONDOWNBLCLK` în cazul în care a înregistrat o succesiune suficient de rapidă de clicuri asupra butonului stâng. Pentru butoanele din mijloc și cel drept sunt generate mesaje corespunzătoare `WM_MBUTTONDOWNBLCLK` și, respectiv, `WM_RBUTTONDOWNBLCLK`. Adăugarea de funcții care să trateze aceste mesaje se face urmând din nou secvența de pași „Adăugarea unei funcții de tratare a mesajului generat de apăsarea butonului mouse-ului”, prezentată în secțiunea precedentă. Puteți adăuga o funcție care să răspundă la efectuarea unui dublu clic asupra butonului stâng pentru a experimenta următoarea secvență de cod:

```
void CMouseMsgDlg::OnLButtonDblClk(UINT nFlags,
    CPoint point)
{
    AfxMessageBox("Left Button Double Clicked");
    CDialog::OnLButtonDblClk(nFlags, point);
}
```

Dacă asamblați și rulați programul după adăugarea funcției de mai sus, la efectuarea unui dublu clic cu butonul stâng al mouse-ului va fi afișată o casetă de mesaj. Ca și în cazul celorlalte mesaje legate de mouse, puteți observa că sunt transmiși aceiași parametri `nFlags` și `point` care vă permit să determinați combinația de taste și de butoane apăsate și, respectiv, poziția în care s-a efectuat dublul clic.

Stabilirea intervalului de dublu clic

Intervalul maxim dintre două clicuri care permite interpretarea acestora ca un dublu clic poate fi stabilit prin intermediul casetei de dialog Mouse, aceasta fiind accesibilă din panoul de control și conținând un glisor prin care se reglează intervalul și o imagine Jack-in-a-Box ce permite testarea valorii stabilite.

Urmărirea deplasării mouse-ului și a poziției acestuia

Aflarea poziției mouse-ului poate fi utilă, dar există situații în care va trebui să urmăriți mouse-ul în timp ce utilizatorul efectuează o operație de tragere. Un program de desenare ar trebui probabil să facă așa ceva pentru a permite utilizatorului să traseze cu mâna liberă. Windows răspunde la această necesitate generând un mesaj la fiecare deplasare a mouse-ului, transmițând de fiecare dată poziția curentă a cursorului mouse-ului.

Limitarea răspunsului la deplasarea mouse-ului

În mod normal, autorii aplicațiilor încearcă să limiteze operațiile efectuate la interceptarea acestui mesaj, deoarece este posibilă transmiterea unui număr mare de mesaje în succesiune rapidă în timp ce utilizatorul deplasează mouse-ul. Un răspuns prea complex va încetini întregul sistem și poate produce discrepante între poziția mouse-ului și operațiile de desenare aferente.

Tratarea evenimentului de deplasare a mouse-ului

Ca în cazul operării butoanelor mouse-ului, deplasarea acestuia este adusă la cunoștința aplicațiilor printr-un nou mesaj, `WM_MOUSEMOVE`. Adăugarea unei funcții de tratare pentru acest mesaj se face la fel cu tratarea butoanelor mouse-ului. Urmăriți secvența de pași „Adăugarea unei funcții de tratare a mesajului generat de apăsarea butonului mouse-ului” prezentată în secțiunea „Tratarea mesajelor generate de apăsarea și eliberarea butoanelor mouse-ului” a acestui capitol, folosind în loc mesajul `WM_MOUSEMOVE`. Vrajitorul va genera pentru dumneavoastră o funcție de tratare `OnMouseMove()`, implementată în listingul 8.2.

LISTINGUL 8.2 LST8_2.CPP – Afișarea și reținerea poziției cursorului în timpul deplasării mouse-ului prin intermediul funcției de tratare `OnMouseMove()`

```
1 void CMouseMsgDlg::OnMouseMove(UINT nFlags,
    CPoint point)
2 {
3     // TODO: Add your message handler code here and/or
4
5     // ** Afisam pozitia mouse-ului
6     CString strMessage;
7     strMessage.Format("Mouse Position = (%d,%d)",
8                         point.x, point.y);
9     SetWindowText(strMessage);
10
11     // ** Retinem punctul si redesenam fereastra
12     m_ptMouse = point;
13     Invalidate(); ❶
14
15     CDialog::OnMouseMove(nFlags, point);
16 }
```

❶ Poziția mouse-ului este reținută într-o variabilă membru pentru a fi utilizată mai apoi în funcția `OnPaint()`.

Remarcați că parametrii transmiși funcției de tratare `OnMouseMove()` sunt aceiași cu cei transmiși funcțiilor de tratare a evenimentelor generate de butoane. Puteți verifica ce butoane sunt apăstate testând `nFlags` în conjuncție cu valorile din tabelul 8.1, exact așa cum se întâmplă în funcția de tratare prezentată în listingul 8.1. Cel de-al doilea parametru, `point`, reprezintă poziția mouse-ului; aceasta este afișată aici pe bara de titlu, în liniile 6-9 ale listingului 8.2. Apelul funcției `Invalidate()` din linia 13 forțează redesenarea ferestrei.

Puteți salva poziția mouse-ului în cadrul clasei `CMouseMsgDlg` sub forma unei variabile membru, așa cum o arată linia 12 a listingului 8.2, cu scopul de a actualiza afișarea casetei de dialog. Odată inserate liniile prezentate în listing, va trebui să adăugați variabila membru `m_ptMouse` la definiția clasei. Aceasta se face ușor, urmând pașii de mai jos pentru adăugarea unei variabile membru în clasa dialog.

Adăugarea unei variabile membru în clasa dialog

1. Efectuați un clic pe pagina **ClassView** din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul + din partea superioară pentru a afișa clasele conținute în proiect.
3. Efectuați un clic cu butonul drept pe clasa `CMouseMsgDlg` pentru a apela meniul de context.
4. Selectați opțiunea **Add Member Variable** pentru a deschide caseta de dialog **Add Member Variable**.
5. Introduceți `CPoint` ca tip al variabilei în caseta **Variable Type**.
6. Introduceți numele variabilei, `m_ptMouse`, în caseta **Variable Declaration**.
Pentru exemplul de față puteți lăsa selectată opțiunea implicită **Public Access** în cadrul grupului **Access Type** (ceea ce permite accesarea variabilei de către orice obiect). Ca alternativă, puteți opta pentru **Protected** sau **Private** în cazul în care nu doriți ca obiecte ale altor clase să aibă acces la noua variabilă membru.
7. Efectuați un clic pe **OK** pentru a adăuga noua variabilă membru.

În acest fel ați adăugat variabila membru `m_ptMouse`, care poate fi utilizată acum în funcția de tratare `OnPaint()` pentru a actualiza aspectul casetei de dialog în funcție de ultima poziție a mouse-ului. În această funcție ați putea să scrieți o secvență de cod care desenează o pereche de ochi ce urmăresc cursorul mouse-ului în deplasarea sa, așa cum ilustrează listingul 8.3. Pentru a adăuga acest cod, efectuați un dublu clic pe funcția membru `OnPaint()` în cadrul paginii **ClassView** a secțiunii spațiului de lucru al proiectului. S-ar putea să fie necesar să efectuați un clic pe semnul plus (+) aflat alături de `CMouseMsgDlg` pentru a afișa funcțiile membru. În funcția noastră există deja cod care desenează pictograma aplicației în cazul în care funcția `IsIconic()` întoarce valoarea `TRUE` (aplicația este minimizată). Ignorați toate acestea și inserați propriul cod între acoladele ramurii `else`, după comentariul `// TODO:`.

LISTINGUL 8.3 LST8_3.CPP – Desenarea unei perechi de ochi care urmăresc poziția cursorului mouse-ului

```

1 else
2 {
3     // ** Cream un context dispozitiv pentru desenare
4     CPaintDC dc(this); — ❶
5
6     // ** Determinam dimensiunea casetei de dialog
7     CRect rcDlg;
8     GetClientRect(&rcDlg);
9
10    // ** Iteram de doua ori, cate o data pentru fiecare ochi
11    for(int i=0;i<2;i++)
12    {
13        // ** Determinam centrul ochiului
14        CPoint ptEye = rcDlg.CenterPoint();
15
16        // ** Stabilim pozitia pentru fiecare ochi
17        ptEye.x += i==0 ? -80 : +80; — ❷
18
19        // ** Cream un dreptunghi pentru ochi
20        CRect rcEye(ptEye,ptEye);
21        rcEye.InflateRect(40,60);
22
23        // ** Desenam ochiul cu alb
24        dc.SelectStockObject(WHITE_BRUSH);
25        dc.Ellipse(rcEye); — ❸
26        rcEye.DeflateRect(20,40);
27
28        // ** Folosim pozitia mouse-ului pentru a desena pupila
29        CPoint ptRel = m_ptMouse - rcEye.CenterPoint();
30        double dX = (double)ptRel.x *
31            (rcEye.Width() / (double)rcDlg.Width());
32        double dY = (double)ptRel.y *
33            (rcEye.Height() / (double)rcDlg.Height()); — ❹
34
35        // ** Ajustam pozitia pupilei si o desenam
36        rcEye.OffsetRect(CPoint((int)dX, (int)dY));
37        dc.SelectStockObject(BLACK_BRUSH);
38        dc.Ellipse(rcEye); — ❺
39    }
40
41    CDialog::OnPaint();
42 }
```

❶ Pentru trasare este inițializat și folosit un context dispozitiv de desenare.

- ❷ Această linie stabilește poziția unui ochi la 80 de pixeli în stânga sau în dreapta față de centrul casetei de dialog.
- ❸ Această funcție `Ellipse()` este folosită pentru a desena fondul alb al ochilor.
- ❹ Poziția curentă a mouse-ului este folosită pentru a calcula poziția relativă a pupilelor în fiecare ochi.
- ❺ Aici se desenează efectiv pupilele, deplasate înspre poziția curentă a mouse-ului.

Linia 4 din listingul 8.3 construiește un obiect context dispozitiv `CPaintDC` pe care îl va folosi pentru desinare. (Despre contextele dispozitiv se discută amănunțit în capitolul 15.) Dimensiunile casetei de dialog sunt determinate prin apelul funcției `GetClientRect()` din linia 8, fiind depuse în obiectul `rcDlg` de tip `RECT`, care a fost declarat în linia 7. `RECT` este o clasă ce conține două obiecte `CPoint`, unul pentru coordonatele colțului din stânga sus al unui dreptunghi și celălalt pentru coordonatele colțului din dreapta jos. Sunt oferite, în plus, o serie de funcții utile pentru determinarea centrului dreptunghiului, a lățimii și a înălțimii și pentru modificarea dimensiunilor acestuia.

Linia 11 este începutul unei bucle care va fi rulată de două ori, câte o dată pentru fiecare ochi. Dreptunghiul care încadrează fiecare ochi este stabilit în linia 14 și deplasat la stânga sau la dreapta liniei mediane în linia 17. Linia 20 fixează dimensiunea adecvată, iar fundalul alb este desenat prin liniile 24-25. Dreptunghiul este apoi ajustat la dimensiuni corespunzătoare pentru pupile în linia 26.

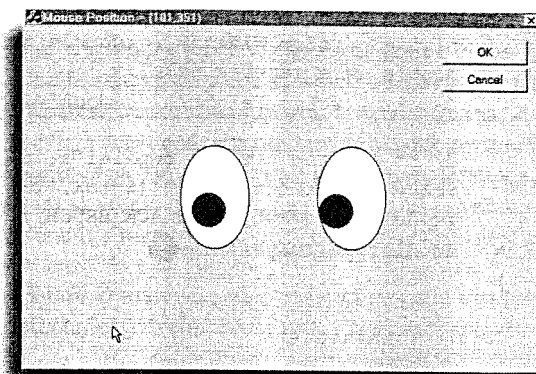
Linia 29 este foarte interesantă. Aici se folosește poziția mouse-ului pentru a calcula poziția pupilei pentru fiecare ochi. Deplasamentul este calculat și stabilit în liniile 30-36, iar desinarea se face în liniile 37 și 38.

Nu am insistat aici cu explicații privind funcțiile de desinare deoarece vom vorbi despre acestea în capitolul 16, „Utilizarea penișelor și a pensulelor”.

Dacă asamblați și rulați aplicația după adăugarea acestui cod, veți avea o casetă de dialog cu doi ochi care urmăresc poziția mouse-ului pe măsură ce-l deplasați în interiorul ferestrei dialogului (vedeți figura 8.2).

FIGURA 8.2

Cei doi ochi urmăresc cursorul mouse-ului în timp ce acesta este deplasat pe suprafața casetei de dialog.



VEDEȚI...

- Pentru mai multe informații privind utilizarea funcției de tratare `OnPaint()`, vedeți pagina 330.
- Pentru a studia desinarea cu ajutorul penișelor și a pensulelor, vedeți paginile 355 și 372.

Capturarea mouse-ului

Dacă ați asamblat aplicația prezentată prin intermediul listingului 8.3, ați remarcat că există o problemă. Totul merge bine până când deplasați cursorul mouse-ului în afara ferestrei dialogului. Motivul este faptul că aplicațiile primesc în mod normal mesajele generate de deplasarea mouse-ului numai atunci când cursorul se află pe suprafața uneia dintre ferestrele componente. În momentul în care mouse-ul depășește marginile ferestrei, mesajele vor fi recepționate de o altă fereastră. Există, însă, o modalitate de a forța transmiterea de mesaje către propria aplicație, tehnică numită *capturarea mouse-ului*.

Reguli privind capturarea mouse-ului

Mouse-ul nu poate fi capturat decât de o fereastră aflată în prim-plan. Dacă o fereastră aflată pe fundal va încerca să captureze mouse-ul, ea va primi mesaje de la mouse numai pentru zonele vizibile din suprafața sa. La un moment dat, mouse-ul nu poate fi capturat decât de o singură aplicație care rulează în sistem.

Capturarea mouse-ului se efectuează simplu, apelând funcția `SetCapture()` din cadrul ferestrei aplicației care urmează să rețină mouse-ul. Trebuie menționat că această operație de capturare este anti-socială față de celelalte aplicații care doresc să răspundă la acțiunile mouse-ului, așa că va trebui să apelați întotdeauna `ReleaseCapture()` pentru a elibera mouse-ul și a permite reluarea unei funcționări normale.

Locuri potrivite pentru capturarea și eliberarea mouse-ului sunt funcțiile care răspund la apăsarea și, respectiv, eliberarea butoanelor. Pornind de la exemplul prezentat în listingul 8.3, modificați funcția de tratare `OnLButtonDown()` pentru a captura mouse-ul, ca mai jos:

```
void CMouseMsgDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
    SetCapture();
    CDialog::OnLButtonDown(nFlags, point);
}
```

Acum fereastra dialogului va captura mouse-ul de fiecare dată când apăsați butonul stâng. Eliberarea mouse-ului se poate face apelând `ReleaseCapture()` în cadrul funcției de tratare `OnLButtonUp()`, ca aici:

```
void CMouseMsgDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    ReleaseCapture();
    CDialog::OnLButtonUp(nFlags, point);
}
```

Asamblați și rulați aplicația după efectuarea acestor modificări. Veți vedea că dacă apăsați și țineți apăsat butonul stâng al mouse-ului în timp ce vă aflați în interiorul ferestrei dialog,

puteți deplasa cursorul oriunde pe ecran și ochii vor continua să-l urmărească. Aceasta înseamnă că aplicația primește în continuare mesaje WM_MOUSEMOVE, chiar și atunci când cursorul nu se află în interiorul ferestrei dialog. De asemenea, observați cum coordonatele devin valori negative atunci când vă situați în stânga și deasupra casetei de dialog, deoarece poziția mouse-ului rămâne relativă la colțul stânga sus al ferestrei.

În momentul în care eliberați butonul mouse-ului este apelată funcția ReleaseCapture(), iar comportamentul ochilor revine la normal, arătând că mouse-ul nu mai este capturat.

Implementarea unui test de clic

O necesitate uzuală legată de libera deplasare a mouse-ului este capacitatea de a testa dacă s-a efectuat clic cu mouse-ul într-o anumită zonă, fără a mai utiliza un control de tip buton. Spre exemplu, dacă vreți să puteți efectua clic pe unul dintre ochii din exemplul nostru pentru a-i schimba culoarea, trebuie să implementați un *test de clic*.

Punctul sensibil al cursorului

Poziția curentă a mouse-ului este reprezentată de fapt printr-un singur pixel particular al cursorului. Acest pixel se numește punct sensibil și se află de obicei în vârful săgeții. Poziția punctului sensibil poate fi specificată la crearea cursorului. Un test de clic reușește numai atunci când acest punct sensibil se află în interiorul dreptunghiului considerat.

Efectuarea unui test de clic nu înseamnă altceva decât a verifica dacă un punct se află în interiorul unui dreptunghi anume. Mai întâi trebuie să rețineți poziția ultimului clic de mouse. Aceasta înseamnă copierea parametrului point transmis funcției de tratare OnLButtonDown() într-o variabilă membru:

```
void CMouseMsgDlg::OnLButtonDown(UINT nFlags, CPoint point)
{
```

```
    SetCapture();
```

```
    m_ptButton = point; // ** Retinem punctul primit
```

Trebuie să adăugați variabila membru m_ptButton de tip CPoint în cadrul clasei CMouseMsgDlg, așa cum ați procedat atunci când urmăream deplasarea mouse-ului, urmând secvența de pași pentru adăugarea unei variabile membru într-o clasă și specificând m_ptButton ca nume al variabilei (tipul rămâne CPoint).

Odată reținut ultimul clic de mouse, puteți verifica dacă m_ptButton se află în interiorul unuia dintre ochi modificând puțin funcția OnPaint() după comentariul // ** Desenam ochiul cu alb, ca aici:

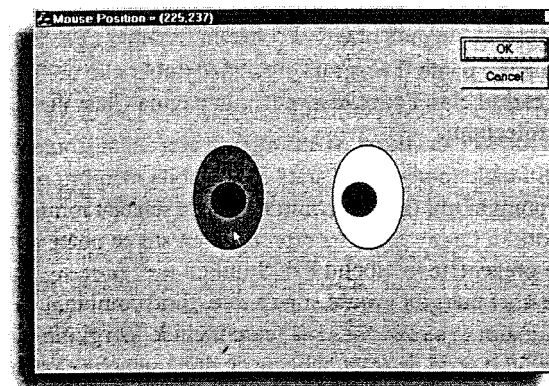
```
// ** Desenam ochiul cu alb
if (rc.Eye.PtInRect(m_ptButton))
    dc.SelectStockObject(GRAY_BRUSH);
else
    dc.SelectStockObject(WHITE_BRUSH);
...
```

Funcția membru PtInRect() a clasei CRect întoarce valoarea TRUE dacă punctul primit ca parametru (în cazul nostru, variabila m_ptButton) se află în interiorul dreptunghiului de test rcEye. Dacă punctul se află într-adevăr în interiorul ochiului, în locul pensulei WHITE_BRUSH este folosită pensula GRAY_BRUSH, ceea ce duce la desenarea ochiului cu culoarea gri.

Asamblați și rulați aplicația după efectuarea acestor modificări și veți vedea că efectuarea unui clic în interiorul unui ochi îl colorează pe acesta cu gri, așa cum o ilustrează figura 8.3. Dacă efectuați apoi clic pe celălalt ochi, acesta va deveni acum gri pentru că punctul se află în interiorul dreptunghiului său. Dacă efectuați un clic altundeva, ambii ochi devin din nou albi, deoarece punctul nu se mai află în nici unul dintre dreptunghiuri.

FIGURA 8.3

Testul de clic colorează un ochi cu gri dacă punctul în care s-a efectuat clic se află pe suprafața ochiului respectiv.



VEDEȚI...

► Pentru a afla mai multe despre pensulele din stoc, vedeți pagina 369.

Utilizarea clasei CRectTracker

Probabil că ați folosit dreptunghiuri de încadrare elastice în unele aplicații pentru a selecta simultan mai multe elemente. Această funcționalitate poartă numele de *încadrare rectangulară* și se poate implementa ușor prin intermediul clasei CRectTracker. Această clasă conține cod care se ocupă de capturarea mouse-ului, permite utilizatorului să selecteze o suprafață și apoi permite aplicației să testeze dacă diferite puncte se află în interiorul dreptunghiului definit de utilizator.

Puteți adăuga un obiect CRectTracker în exemplul nostru pentru a selecta unul sau ambii ochi prin trasarea unui cadru elastic în jurul suprafeței de selecție definite, testând apoi dacă ochii se află în interiorul dreptunghiului stabilit.

Pentru a adăuga în clasa CMouseMsgDlg o variabilă membru numită m_RectTracker și având tipul CRectTracker, urmați secvența de pași pentru adăugarea unei variabile membru în cadrul clasei dialog.

Primul lucru care trebuie făcut este să inițializați obiectul de încadrare cu un dreptunghi inițial și un parametru de stil. În acest sens puteți să inserați codul de mai jos în funcția constructor `CMouseMsgDlg::CMouseMsgDlg()`, fiind rulat astfel la inițializarea clasei `CMouseMsgDlg`:

```
CMouseMsgDlg::CMouseMsgDlg(CWnd* pParent /*=NULL*/)
: CDialog(CMouseMsgDlg::IDD, pParent),
  m_RectTracker(CRect(0,0,0,0),
  // ** Dreptunghiul initial
  CRectTracker::hatchedBorder +
  // ** Margine hasurata
  CRectTracker::resizeOutside)
  // ** Butoane de dimensionare exterioare
```

Această funcție poate fi accesată efectuând dublu clic pe funcția membru `CMouseMsgDlg()` afișată în cadrul clasei `CMouseMsgDlg`, în pagina `ClassView` din secțiunea spațiului de lucru al proiectului.

Liniiile adiționale (comentate) arată că variabila membru `m_RectTracker` este inițializată cu un dreptunghi nul (astfel că inițial nu este selectat nimic). Al doilea parametru trimis constructorului `CRectTracker` reprezintă un stil ce poate obținut prin combinația valorilor simbolice prezentate în tabelul 8.4. Stilul `CRectTracker::hatchedBorder` are ca efect desenarea unei margini groase și hașurate, fiind combinat cu `CRectTracker::resizeOutside`, care include marginile în suprafața de selecție (se elimină astfel o serie de teste delicate de după selecție care ar fi fost necesare dacă dreptunghiul selectat era prea mic).

TABELUL 8.4 Valori de stil pentru *CRectTracker*

Valoare de stil	Descriere
<code>CRectTracker::solidLine</code>	Trasează cadrul elastic cu linie continuă.
<code>CRectTracker::dottedLine</code>	Trasează cadrul elastic cu linie întreruptă.
<code>CRectTracker::hatchedBorder</code>	Trasează cadrul elastic cu o linie groasă, hașurată.
<code>CRectTracker::hatchInside</code>	Desenează o hașură peste suprafața selectată.
<code>CRectTracker::resizeInside</code>	Marginea de dimensionare se consideră a fi în interiorul dreptunghiului.
<code>CRectTracker::resizeOutside</code>	Marginea de dimensionare se consideră a fi în exteriorul dreptunghiului.

Operația de încadrare elastică începe de obicei cu apăsarea de către utilizator a butonului stâng al mouse-ului, așa că funcția de tratare `OnLButtonDown()` este locul evident pentru declanșarea acestei operații. Întregul proces de încadrare elastică este realizat de funcția membru `TrackRubberBand()` a clasei `CRectTracker`. În funcția `OnLButtonDown()` veți adăuga cod ca aici:

```
void CMouseMsgDlg::OnLButtonDown(UINT nFlag, CPoint point)
{
    m_RectTracker.TrackRubberBand(this, point, TRUE);
    CDialog::OnLButtonDown(nFlags, point);
}
```

Precum vedeți, funcția `TrackRubberBand()` necesită trei parametri. Primul dintre aceștia este un pointer la fereastra în care va avea loc operația de încadrare elastică. Cum noi vrem ca această fereastră să fie caseta de dialog, vom transmite ca parametru pointerul special `this`, care indică însuși obiectul `CMouseMsgDlg` (care este derivat din clasa `CWnd` și, prin urmare, este un obiect de tip fereastră).

Al doilea parametru reprezintă punctul de început pentru colțul din stânga sus al dreptunghiului elastic de încadrare. Aici putem transmite poziția în care s-a efectuat clic.

Cel de-al treilea parametru este opțional și poate fi stabilit la `TRUE`, pentru a permite extinderea dreptunghiului de încadrare în stânga și deasupra poziției de start, sau poate fi `FALSE`, pentru ca utilizatorul să poată extinde cadrul numai în jos și la dreapta față de poziția de început.

Mai devreme, funcția de tratare `OnLButtonDown()` conținea un apel `SetCapture()`, care acum a fost înlăturat pentru că obiectul de încadrare rectangulară se va ocupa de capturarea mouse-ului, caz în care propriul apel `SetCapture()` ar interfera. Acum trebuie să eliminați și apelul `ReleaseCapture()` corespunzător din funcția de tratare `OnLButtonUp()`:

```
void CMouseMsgDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    CDialog::OnLButtonUp(nFlags, point);
}
```

În fine, trebuie să modificați testul de clic din secvența de desenare a funcției `OnPaint()` pentru a înlocui verificarea membrului de tip punct cu un test asupra obiectului de încadrare. Clasa `CRectTracker` include o funcție de verificare proprie, `HitTest()`, pe care o puteți apela transmițând obiectul `CPoint` pe care doriți să-l testați:

```
// ** Desenam ochiul cu alb
if (m_RectTracker.HitTest(rcEye.CenterPoint()))
    !=CRectTracker::hitNothing)
    dc.SelectStockObject(GRAY_BRUSH);
else
    dc.SelectStockObject(WHITE_BRUSH);
...
```

Transmițând `rcEye.CenterPoint()` în apelul `HitTest()` verificați dacă centrul dreptunghiului asociat fiecărui ochi se află în interiorul dreptunghiului de selecție. `HitTest()` întoarce `CRectTracker::hitNothing(-1)` dacă punctul nu se află în interiorul suprafeței selectate. Dacă punctul se încadrează în suprafața selectată, este întoarsă una din mai multe valori prin care se indică poziția exactă a punctului relativ la zona selectată. Aceste valori sunt prezentate mai jos (vedeți tabelul 8.5), dar valorile

întâlnite în mod normal sunt `CRectTracker::hitMiddle` dacă punctul se află în interiorul suprafeței și `CRectTracker::hitNothing` în caz contrar.

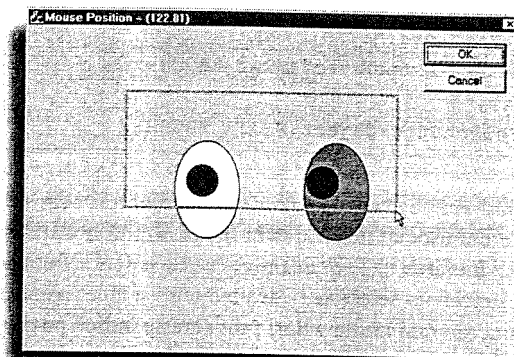
TABELUL 8.5 Valorile întoarse de funcția *HitTest()* a clasei *CRectTracker*

Valoarea întoarsă	Valoare	Semnificație
<code>CRectTracker::hitNothing</code>	-1	Punctul nu se află în interiorul dreptunghiului selectat
<code>CRectTracker::hitTopLeft</code>	0	Punctul se află în colțul stânga sus al dreptunghiului
<code>CRectTracker::hitTopRight</code>	1	Punctul se află în colțul dreapta sus al dreptunghiului
<code>CRectTracker::hitBottomRight</code>	2	Punctul se află în colțul dreapta jos al dreptunghiului
<code>CRectTracker::hitBottomLeft</code>	3	Punctul se află în colțul stânga jos al dreptunghiului
<code>CRectTracker::hitTop</code>	4	Punctul se află pe latura superioară a dreptunghiului
<code>CRectTracker::hitRight</code>	5	Punctul se află pe latura dreaptă a dreptunghiului
<code>CRectTracker::hitBottom</code>	6	Punctul se află pe latura inferioară a dreptunghiului
<code>CRectTracker::hitLeft</code>	7	Punctul se află pe latura stângă a dreptunghiului
<code>CRectTracker::hitMiddle</code>	8	Punctul se află undeva în interiorul dreptunghiului

Dacă asamblați și rulați programul în urma efectuării acestor modificări, veți putea să creați un dreptunghi de selectare apăsând butonul stâng al mouse-ului, după care-l veți putea extinde la dimensiunile dorite în timp ce țineți butonul apăsat. Dacă centrul unui ochi se află în interiorul dreptunghiului, ochiul respectiv va fi selectat și colorat cu gri, așa cum se vede în figura 8.4.

FIGURA 8.4

Utilizarea clasei `CRectTracker` pentru selectarea prin cadru elastic.



CAPITOLUL

9

Utilizarea controalelor ActiveX

Adăugarea în proiect a unor controale ActiveX mai sofisticate

Personalizarea proprietăților și stilurilor controalelor

Utilizarea în program a datelor furnizate de controale

Întreaga evoluție a programării din ultimul deceniu s-a concentrat în general asupra rezolvării sindromului de reinventare a roții, care a obsedat lumea programatorilor în anii '80. Tehnici precum *orientarea pe obiect* și *COM/OLE* (Component Object Model/Object Linking and Embedding – Modelul componentelor obiectuale/Legarea și încapsularea obiectelor) reprezintă culmi ale eforturilor întreprinse. În prezent este posibilă achiziționarea unor *controale ActiveX* foarte sofisticate care pot fi integrate repede și ușor în aplicații, economisind ore, săptămâni sau chiar luni de muncă de programare. Mai multe controale sunt distribuite gratuit sau disponibile ca shareware. Dacă doriți să creați controale, foarte bine; le puteți pune la dispoziție gratuit sau le puteți vinde altor programatori. Una peste alta, toată lumea profită deoarece aplicațiile se pot dezvolta mai repede, mai ieftin și cu mai multe facilități.

Controalele ActiveX și World Wide Web

Odată cu apariția paginilor de server active, controalele ActiveX și scripturile de tip server pot fi adăugate în paginile de Web de pe Internet în scopul realizării unor aplicații sofisticate de Web.

VEDEȚI...

- În legătură cu crearea controalelor ActiveX, vedeți pagina 605.
- Pentru a afla mai multe despre COM și OLE, vedeți pagina 581.

Selectarea și adăugarea controalelor ActiveX din galeria de componente

Galeria de componente reprezintă o colecție de componente care pot fi integrate în propriile aplicații în scopul rafinării acestora. Visual C++ este însoțit de o gamă largă de controale ActiveX disponibile imediat, iar această mulțime poate fi îmbogățită (în orice moment) cu noi controale. Controalele ActiveX se bazează pe tehnologia COM, ceea ce înseamnă că pot fi utilizate în Visual C++, Visual Basic, Visual J++ și în orice alte medii de programare care pot comunica cu obiecte COM. Din această cauză, în documentație apar de multe ori exemple din Visual Basic – nu vă lăsați descurajat, pentru că aceleași metode sunt disponibile și aplicațiilor Visual C++.

Inspectarea controalelor ActiveX

Puteți crea o aplicație de tip dialog pentru a experimenta adăugarea și utilizarea controalelor ActiveX, acestea putând fi plasate direct pe caseta de dialog principală a aplicației. Apelați la AppWizard pentru a crea un nou proiect de tip dialog numit ActiveX (urmărind pașii prezentați în capitolul 3, „Crearea și proiectarea casetelor de dialog“).

Din interiorul proiectului puteți apela o listă cu controalele ActiveX înregistrate în sistem urmând pașii de mai jos:

Inspectarea controalelor ActiveX din galeria de componente

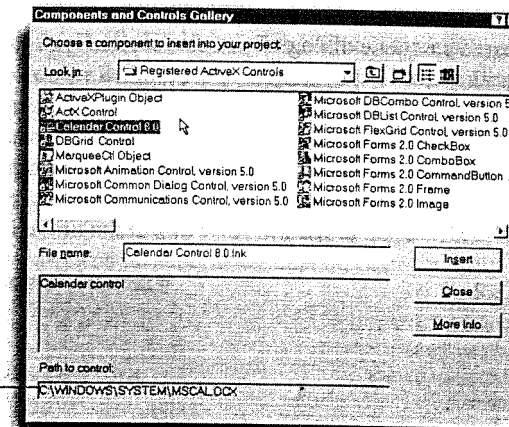
1. Efectuați un clic pe meniul **Project** și selectați submeniul **Add to Project**.

2. Selectați opțiunea **Components and Controls** din submeniul deschis.
3. Efectuați un dublu clic pe catalogul **Registered ActiveX Controls** din caseta de dialog Components and Controls Gallery.
4. Ar trebui să vedeți acum o listă cu controalele ActiveX instalate în sistem, asemeni celei din figura 9.1.

FIGURA 9.1

Controalele ActiveX înregistrate sunt afișate în galeria de componente.

- 1 Codul executabil pentru controlul Calendar ActiveX se află în acest fișier .ocx. (Controlul Calendar însoțește Office 97, iar dumneavoastră s-ar putea să aveți o versiune mai veche, mai recentă sau chiar nici o versiune, totul în funcție de programele care ar fi putut să instaleze acest control.)



Observați cum controalele din caseta de dialog Components and Controls Gallery sunt afișate ca scurtături obișnuite din Windows Explorer. Fișierele care conțin codul pentru fiecare control sunt afișate în caseta **Path to Control** din partea inferioară a ferestrei. Codul unui control ActiveX este plasat de fapt într-un fișier .OCX sau .DLL, așa că dacă intenționați distribuirea unei aplicații va trebui să furnizați și aceste fișiere asociate și să le înregistrați în sistemul beneficiar. Înregistrarea controalelor se poate face cu ajutorul programului regsvr32.exe, apelându-l fie prin opțiunea **Run** din meniul **Start**, fie din linia de comandă, ca aici:

```
regsvr32 C:\Windows\System\mscal.ocx
```

Comanda de mai sus are ca efect înregistrarea controlului Calendar ActiveX și este necesară numai la distribuirea sau instalarea de controale noi; controalele pe care le vedeți în galeria de componente sunt deja înregistrate în sistem.

Dacă selectați un control, este posibil ca butonul **More Info** să fie activat. În acest caz puteți efectua un clic pe acest buton pentru a afișa o fereastră standard de asistență care prezintă controlul și modul de utilizare a acestuia.

În cadrul paginilor de asistență veți vedea că, asemeni controalelor obișnuite, controalele ActiveX dispun de mai multe proprietăți, metode și evenimente. Proprietățile vă permit să modificați aspectul controlului. Metodele vă permit să fixați sau să determinați datele conținute în control. Evenimentele apelează codul pe care-l scrieți în momentul în care utilizatorul interacționează cu respectivul control (de exemplu, efectuarea unui clic pe un buton).

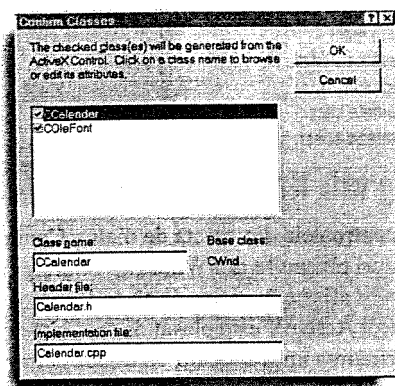
Inserarea de noi controale în proiectul curent

Pentru a adăuga controlul Calendar într-o aplicație de tip dialog, efectuați un clic pe scurtătura **Calendar Control 8.0** care ar trebui să figureze în partea superioară a listei controalelor instalate. Odată selectat, controlul Calendar poate fi inserat în aplicație prin efectuarea unui clic pe butonul **Insert**. O casetă de dialog vă va cere confirmarea; efectuați un clic pe **OK** pentru a insera controlul.

Următoarea casetă de dialog afișată este caseta **Confirm Classes**, ilustrată în figura 9.2. Aceasta vă aduce la cunoștință numele fișierelor care vor fi adăugate la proiect pentru a implementa clasele de interfață (numite și *clase dispecer*) necesare controlului. Dacă este nevoie, aici puteți să modificați numele claselor de interfață și ale fișierelor. În mod normal veți accepta denumirile propuse, dar uneori acestea ar putea intra în conflict cu clase și fișiere pe care le utilizați deja. Puteți, de asemenea, să eliminați unele dintre clasele propuse pentru adăugare, dar de cele mai multe ori veți insera toate clasele prezentate.

FIGURA 9.2

Casetă de dialog **Confirm Classes** vă permite să modificați numele fișierelor de implementare.



Dacă nu modificați opțiunile implicite și efectuați un clic pe **OK**, noile clase și fișierele de implementare corespunzătoare vor fi adăugate în proiect. Puteți vedea aceste clase închizând caseta de dialog **Components and Controls Gallery** și selectând pagina **ClassView** din secțiunea spațiului de lucru al proiectului. Cele două noi clase, **CCalendar** și **COleFont**, conțin funcții de interfațare pentru controlul Calendar și pentru fontul utilizat de acesta.

Ce sunt aceste funcții de interfațare? Dacă efectuați un clic pe semnul plus din dreptul clasei **CCalendar** pentru a afișa funcțiile membru ale acesteia și apoi efectuați un dublu clic pe funcția **GetBackColor()**, veți vedea în fereastra de editare că implementarea funcției nu conține decât câteva linii, ca aici:

```
unsigned long CCalendar::GetBackColor()
{
    unsigned long result;
    InvokeHelper(DISPID_BACKCOLOR, DISPATCH_PROPERTYGET,
        VT_I4, (void*)&result, NULL);
}
```

```
return result;
}
```

Celelalte funcții sunt similare cu cea de mai sus. Acestea sunt *funcții de interfațare*, sau *funcții dispecer*. Ele pot fi apelate ca orice funcție, dar adevărata implementare se află în fișierul **.OCX** sau **.DLL** asociat controlului. Aceste funcții de interfațare utilizează o interfață **COM** pentru a executa codul propriu-zis, apelând la funcția **InvokeHelper()**.

Parametrii trimiși unei funcții sau preluați prin intermediul acesteia sunt convertiți la un tip generic (**Variant**) și apoi trimiși către sau de către codul propriu-zis. Astfel sunt atinse trei scopuri. În primul rând, dimensiunea aplicației este redusă, deoarece este necesar doar codul acestor funcții de interfațare. În al doilea rând, codul pe care-l scrieți devine în mod automat compatibil cu versiunile ulterioare ale controlului folosit (presupunând că autorul controlului păstrează interfața originală), astfel că utilizatorii pot actualiza doar controalele, fără a mai avea nevoie de o nouă versiune a aplicației. În al treilea rând, este protejată proprietatea intelectuală a autorului controlului, deoarece codul sursă al acestuia nu este accesibil.

Controalele din Visual Basic și de tip VBX și compatibilitatea

Poate că ați auzit de **VBX**; acestea sunt controale Visual Basic, fiind predecesorii controalelor **OCX** și, deși operează diferit față de acestea din urmă, pot fi utilizate împreună cu compilatorul **Visual C++ pe 16 biți**. În schimb, controalele **VBX** nu mai sunt compatibile cu **Visual C++ pe 32 de biți** sau cu programele pe 32 de biți – în cadrul aplicațiilor moderne pe 32 de biți trebuie să utilizați controale **OCX** (controale **ActiveX**).

VEDEȚI...

➤ Pentru a afla mai multe despre **COM** și **OLE**, vedeți pagina 581.

Selectarea, dimensionarea și testarea controalelor ActiveX pornind de la paleta de controale

Pe lângă faptul că la proiect au fost adăugate clasele de interfață pentru noul control, în paleta de controale din editorul de resurse a apărut o nouă pictogramă care reprezintă controlul **ActiveX** pe care l-ați importat.

Adăugarea unui control ActiveX într-o casetă de dialog

Noul control **Calendar** poate fi utilizat într-o casetă de dialog. Selectați pagina **ResourceView** și expandați resursele proiectului **ActiveX** pentru a le putea vedea. Apoi deschideți catalogul **Dialog** și efectuați un dublu clic pe **IDD_ACTIVEX_DIALOG** pentru a edita caseta de dialog principală a aplicației. Măriți dimensiunea acestei casete de dialog și înlăturați eticheta **TODO: Place dialog controls here**.

În paleta de controale ar trebui să apară acum la sfârșit o pictogramă nouă, corespunzătoare controlului **Calendar**, așa cum se vede în figura 9.3.

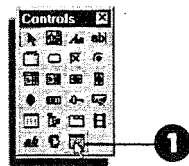
Puteți să selectați și să plasați acest control pe suprafața casetei de dialog la fel cum ați face-o cu orice alt control. Odată adăugat controlul pe caseta de dialog, puteți să-i măriți dimensiunile ca în cazul unui control oarecare. Extindeți controlul astfel încât să fie aliniat

cu laturile superioară și stângă ale casetei de dialog. S-ar putea să fie necesar să măriți dimensiunea casetei de dialog, deoarece controalele Calendar sunt relativ mari.

FIGURA 9.3

Noul control Calendar ActiveX adăugat în paleta de controale.

- ❶ Pictograma controlului Calendar 8.0



Adăugarea de controale ActiveX prin intermediul editorului de dialoguri

În loc să selectați controale ActiveX din galeria de componente, în cazul în care știți exact ce control doriți există o scurtătură la această operație, disponibilă în cadrul editorului de dialoguri. Prin efectuarea unui clic cu butonul drept al mouse-ului pe suprafața machetei de dialog deschisă în editor veți apela meniul de context al editorului. Selectați opțiunea **Insert ActiveX Control** a acestui meniu pentru a afișa caseta de dialog Insert ActiveX Control.

Această casetă de dialog afișează o listă cu controalele ActiveX instalate, iar de aici puteți selecta direct un control pentru a-l adăuga în macheta de dialog deschisă. Alegeți din listă controlul **Microsoft MCI Control, version 5** (rețineți că numărul de versiune ar putea diferi). La efectuarea unui clic pe **OK** ar trebui să fie afișat noul control. Controlul MCI (Media Control Interface – interfață pentru controlul multimedia), care la momentul proiectării casetei de dialog este reprezentat printr-un dreptunghi alb, trebuie plasat înspre colțul stânga jos al noii casete de dialog.

Utilizarea acestei metode nu adaugă și interfața pentru control, dar această operație se poate efectua prin intermediul ClassWizard, odată cu maparea peste control a unei variabile membru, așa cum veți vedea în secțiunea „Adăugarea unei variabile membru de tip clasă dispecer pentru un control”, mai târziu în acest capitol.

Controlul MCI pune la dispoziție un set de butoane care pot fi conectate la un dispozitiv multimedia, așa cum sunt dispozitivul waveaudio pentru redarea fișierelor de sunet .WAV, dispozitivul cdaudio pentru redarea CD-urilor sau dispozitivul digitalvideo pentru redarea video-clipurilor.

Testarea controalelor în cadrul editorului de dialoguri

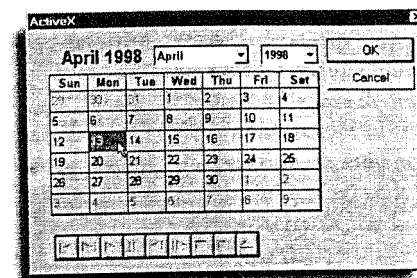
Aceste controale pot fi testate folosind operația uzuală de testare din cadrul editorului de dialoguri. În acest sens, efectuați un clic pe meniul **Layout** și selectați opțiunea **Test** (sau apăsați Ctrl+T). Ar trebui să vedeți controalele ActiveX așa cum arată acestea de fapt (vedeți figura 9.4) și să puteți interacționa cu ele.

VEDEȚI...

- Pentru mai multe detalii privind editarea casetelor de dialog, vedeți pagina 54.
- Pentru a afla mai multe despre MCI și despre utilizarea bibliotecilor API, vedeți pagina 681.

FIGURA 9.4

Testarea casetei de dialog conținând noile controale ActiveX.



Modificarea proprietăților controalelor în cadrul editorului de resurse

Ca și în cazul controalelor obișnuite, pentru controalele ActiveX pot fi stabilite o serie de proprietăți, acestea fiind grupate pe pagini într-un catalog de proprietăți. Proprietățile sunt disponibile atât prin cod, oferite de către funcțiile clasei de interfață la maparea peste control a unei variabile membru, cât și prin opțiunile prezente în fiecare dintre paginile de proprietăți, implementate tot de către control. Aceste pagini de proprietăți pot fi accesate selectând controlul și apăsând Alt+Enter sau efectuând clic pe opțiunea **Properties** din meniul contextual.

Stabilirea proprietăților standard

Și de această dată există o pagină **General**. Ca în cazul controalelor obișnuite, în cadrul acestei pagini puteți să selectați un ID care să identifice în mod unic controlul și să stabiliți opțiunile tipice de activare și vizibilitate. Efectuați un clic cu butonul drept pe controlul Calendar și selectați **Properties Calendar Object** pentru a afișa proprietățile acestuia; apoi înlocuiți identificatorul implicit IDC_CALENDAR1 cu IDC_CALENDAR. Repetați operația pentru controlul MCI de la Microsoft, stabilindu-i identificatorul IDC_MMCONTROL.

Utilizarea paginilor de proprietăți ale controalelor

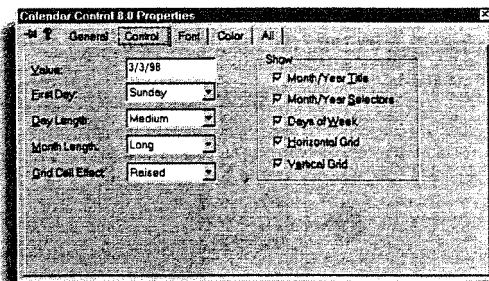
Celelalte pagini de proprietăți sunt specifice controlului în cauză. Apelați din nou proprietățile controlului Calendar și selectați pagina **Control**. Acum puteți vedea o serie de proprietăți specifice controlului, așa cum o arată figura 9.5.

Aceste proprietăți vă permit să schimbați aspectul și stilul controlului Calendar. Încercați să le modificați și urmăriți efectul asupra controlului afișat (s-ar putea să fiți nevoit să deplasați caseta de dialog Properties, astfel încât să nu acopere controlul). Observați că atributele de afișare ale controlului Calendar vă oferă o flexibilitate satisfăcătoare.

Acum selectați pagina **Font**. Observați că aici aveți posibilitatea de a stabili dimensiunile și tipurile de caracter pentru fonturile utilizate de către control. Dacă accesați pagina **Color** veți vedea că și culorile pot fi personalizate.

FIGURA 9.5

Proprietățile controlului Calendar.

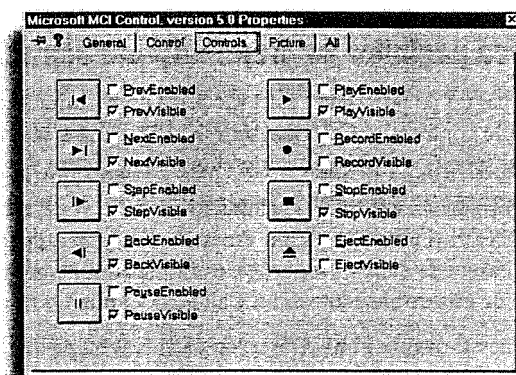


Ultima dintre pagini este **All**. Este o pagină specifică tuturor controalelor ActiveX și conține o listă cu toate proprietățile controlului, împreună cu valorile curente ale acestora. Lista este generată pe baza proprietăților OLE expuse de către control, acestea fiind disponibile și în cadrul clasei controlului, prin intermediul funcțiilor **Set** și **Get** corespunzătoare fiecăreia. În consecință, proprietățile **BackColor** îi corespund funcțiile **GetBackColor()** și **SetBackColor()**, pe care le puteți folosi pentru a afla sau a determina prin cod valoarea asociată.

Închideți proprietățile controlului Calendar, efectuați un clic cu butonul drept pe controlul MCI și selectați **Properties MMControl Object** pentru a afișa proprietățile acestuia din urmă. Dacă inspectați diferitele pagini, veți observa că paginile **General** și **All** sunt destul de similare între controale, în timp ce paginile specifice se deosebesc foarte mult. Dacă selectați pagina **Controls** a controlului MCI veți vedea că butoanele acestuia pot fi activate și afișate individual. Deselectați opțiunile **EjectVisible** și **RecordVisible** (vedeți figura 9.6). Închideți catalogul de proprietăți și apoi apăsați **Ctrl+T** pentru a testa din nou caseta de dialog. Ar trebui să remarcați că cele două butoane au dispărut, iar caseta albă afișată la editarea machetei este corespunzător mai scurtă.

FIGURA 9.6

Pagina **Controls** a controlului MCI de la Microsoft vă permite să selectați individual butoanele vizibile.



Utilizarea claselor asociate controalelor

Până acum ați făcut destule lucruri cu controalele ActiveX fără a adăuga nici o linie de cod, și chiar ați putea să asamblați și să rulați în acest moment aplicația de test pentru a afișa aceste controale și a interacționa cu ele. Ca și în cazul controalelor obișnuite, veți avea nevoie să transferați date înspre și dinspre controale prin cod. În mod normal ați mapa peste un control o variabilă membru având ca tip clasa controlului și ați adăuga-o în clasa casetei de dialog. Lucrurile stau la fel și în cazul controalelor ActiveX, iar la adăugarea codului necesar puteți beneficia de ajutorul oferit de **ClassWizard**.

Adăugarea unei variabile membru de tip clasă dispecer pentru un control

ClassWizard poate fi apelat din editorul de resurse, fie apăsând **Ctrl+W**, fie efectuând un clic pe meniul **View** și selectând opțiunea **ClassWizard**. Accesați pagina **Member Variables** a casetei de dialog **MFC ClassWizard** pentru a afișa o listă cu identificatorii controalelor care pot fi mapate.

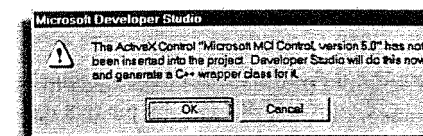
Veți vedea că identificatorii **IDC_CALENDAR** și **IDC_MMCONTROL** din caseta de dialog pot fi mapați în cadrul clasei **CActiveXDlg** a casetei de dialog. Dacă selectați **IDC_CALENDAR** și efectuați un clic pe butonul **Add Variable**, va fi afișată obișnuita casetă de dialog **Add Member Variable**.

Peste controalele ActiveX pot fi mapate numai variabile având ca tip clasa dispecer sau interfață a controlului în cauză. Prin urmare, caseta combinată **Category** conține exclusiv opțiunea **Control**, iar tipul unei variabile pentru controlul Calendar nu poate fi decât **CCalendar**.

Specificați **m_Calendar** ca nume al variabilei membru și efectuați un clic pe **OK** pentru a o adăuga. Selectați identificatorul **IDC_MMCONTROL** și efectuați din nou un clic pe butonul **Add Variable**. De această dată veți vedea o casetă de dialog care vă informează că nu a avut loc adăugarea controlului MCI la proiect, propunându-vă acest lucru (vedeți figura 9.7).

FIGURA 9.7

Propunerea **ClassWizard** de a crea încapsularea C++ pentru ActiveX.



Efectuați un clic pe **OK** și va fi afișată aceeași casetă de dialog **Confirm Classes** care a apărut și atunci când ați selectat controlul Calendar din galeria de componente, datele fiind corespunzătoare acum controlului MCI. Și de această dată puteți modifica, dacă este nevoie, clasele inserate și fișierele care le implementează.

După ce efectuați clic pe butonul **OK** al acestei casete de dialog, veți continua cu caseta de dialog **Add Member Variable**, care vă permite să mapați o variabilă membru de tip **Cmci** în

cadru clasei dialog. Introduceți `m_MCI` în câmpul **Member Variable Name** și efectuați clic pe **OK** pentru a adăuga noua variabilă membru.

Acum ați mapat variabile membru peste controalele din caseta de dialog, așa că puteți să apelați prin cod funcțiile oferite de acestea pentru a citi și a înscrie valori.

Citirea și scrierea prin cod a proprietăților controalelor

Acum puteți adăuga în funcția `OnInitDialog()` a casetei de dialog cod care să inițializeze controalele `ActiveX`, așa cum se vede în listingul 9.1. Controlul `Calendar` dispune de o funcție `Today()` care îl aduce la data curentă. Controlul `MCI` poate fi configurat pentru a reda un fișier `.WAV` ca și clip audio.

LISTINGUL 9.1 LST9_1.CPP – Inițializarea controalelor `ActiveX` `Calendar` și `MCI` în `OnInitDialog()`

```
1 // TODO: Add extra initialization here
2
3 // ** Aducem calendarul la ziua de astazi
4 m_Calendar.Today();
5
6 // ** Configuram dispozitivul mci waveaudio pentru a reda
7 // ** fisierul The Microsoft Sound.wav
8 m_MCI.SetNotify(FALSE);
9 m_MCI.SetWait(TRUE);
10 m_MCI.SetShareable(FALSE);
11 m_MCI.SetDeviceType("waveaudio");
12 m_MCI.SetFileName("C:\\Windows\\Media\\TheMic~1.wav"); // ❶
13 m_MCI.SetCommand("Open");
14
15 return TRUE; // return TRUE unless you set the focus
16 }
```

❶ Controlul `MCI` poate fi asociat cu un fișier de tip clip media, astfel că butoanele sale controlează redarea, oprirea și întreruperea clipului.

În listingul 9.1, controlul `ActiveX` `Calendar` este adus la ziua curentă prin apelul funcției de interfațare `Today()`. Controlul `MCI` este un pic mai delicat de configurat, deoarece poate fi utilizat împreună cu mai multe dispozitive și, prin urmare, dispune de mai multe opțiuni. Liniile de la 8 la 13 configurează controlul `MCI` pentru redarea prin dispozitivul `waveaudio` a fișierului `.WAV` numit `The Microsoft Sound`. Observați că numele de fișier specificat în linia 12 conține câte două caractere backslash pentru a reprezenta un backslash de delimitare a directoroarelor. Aceasta se întâmplă deoarece caracterul backslash are un rol special în `C++`, fiind folosit pentru specificarea altor caractere de control. Așadar, două caractere backslash sunt folosite pentru a reprezenta un singur backslash. Numele fișierului a fost redus la numele scurt din DOS, `TheMic~1.wav`. Dacă lansați `Windows Explorer`, veți putea găsi acest fișier în directorul `C:\\Windows\\Media`, sub numele `The Microsoft Sound`.

Mai este un singur lucru de adăugat. Dispozitivul multimedia `waveaudio` deschis în linia 13 din listingul 9.1 trebuie închis pentru o bună administrare a resurselor sistemului. Cel mai bun moment pentru această operație de închidere este distrugerea casetei de dialog. Cu ajutorul secvenței de pași de mai jos puteți adăuga funcția membru virtuală

`DestroyWindow()`. ← *before destroy waveaudio*

Numele de directorate în `Windows NT`

Aveți grijă în `Windows NT`, unde fragmentul de cale `C:\\Windows` trebuie înlocuit cu `C:\\WINNT`, deoarece acesta este directorul în care sunt plasate implicit fișierele `Windows`. Dacă nu reușiți să găsiți fișierul propus cu `Explorer`, orice alt fișier `.WAV` este la fel de bun, cu condiția ca numele și calea de directorate să fie corecte.

Adăugarea unei funcții virtuale în clasa dialogului

1. Efectuați un clic pe pagina `ClassView` din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus din extremitatea superioară pentru a afișa clasele proiectului.
3. Efectuați un clic cu butonul drept pe clasa `CActiveXDlg` pentru a afișa meniul contextual.
4. Selectați opțiunea **Add Virtual Function** pentru a afișa caseta de dialog `New Virtual Override`. În lista **New Virtual Functions** ar trebui să figureze o funcție `DestroyWindow`.
5. Efectuați un clic pe butonul **Add and Edit** pentru a începe editarea noii funcții.

Această funcție va fi apelată în momentul în care caseta de dialog este distrusă; aici puteți insera un nou apel `SetCommand()` către obiectul control `ActiveX` `m_MCI` cu scopul de a închide dispozitivul `waveaudio`, ca mai jos:

```
BOOL CActiveXDlg::DestroyWindow()
{
    // TODO: Add your specialized code here and/or call the
    m_MCI.SetCommand("Close");
    return CDialog::DestroyWindow();
}
```

Dispozitivul `MCI` va fi închis în mod corespunzător, așa că acum puteți să asamblați și să rulați aplicația. Observați că în controlul `Calendar` este selectată acum data curentă (dacă data din sistem este corectă) și că butonul de redare (pictograma săgeată >) este activat. La efectuarea unui clic pe acest buton ar trebui să auziți clipul audio `Microsoft`. Pe parcursul redării, butonul de întrerupere va fi activat, permițându-vă să întrerupeți și apoi să continuați redarea. Odată încheiată redarea clipului, nu-l puteți relua decât după efectuarea unui clic pe butonul de derulare înapoi.

Dacă aruncați o privire asupra funcțiilor membru ale claselor `ActiveX` `CCalendar` și `Cmci` veți vedea că există o multitudine de alte funcții ce pot fi apelate. De asemenea, nu uitați că există disponibile informații de asistență privind aceste controale, pe care le puteți accesa efectuând clic pe butonul **More Info** din caseta de dialog `Component and Controls Gallery`.

Crearea controalelor ActiveX în cadrul clasei dispecer

Dacă priviți definițiile claselor `CCalendar` și `Cmci` (aflate în fișierele `Calendar.h` și `mci.h`) veți observa că ambele sunt derivate din clasa `CWnd`. Aceasta permite aplicației container să le trateze ca pe orice alte ferestre (ca pe alte controale). Partea inteligentă se află în funcția `Create()`, care este definită în clasa `CWnd` ca funcție virtuală și este supradefinită în clasele dispecer pentru a apela funcția `CreateControl()` a clasei `CWnd`, aceasta creând de fapt un control OLE (ActiveX), și nu o fereastră obișnuită.

Acestea sunt numai două din multele controale oferite în galeria de componente; multe altele pot fi descărcate de pe Internet sau sunt folosite chiar în paginile de Web. Probabil că acum începeți să realizați puterea aflată în spatele acestui cod simplu – conceptul de refolosire pus în practică de COM.

Implementarea funcțiilor de tratare a evenimentelor ActiveX prin intermediul ClassWizard

În mod evident, controalele ActiveX ar fi destul de limitate dacă nu ați putea primi mesaje care să indice că utilizatorul a modificat starea unui control. Din fericire, aceste controale au și capacitatea de a genera evenimente în cadrul aplicațiilor. Adăugarea funcțiilor de tratare a evenimentelor generate de aceste controale se face în maniera cunoscută, prin intermediul ClassWizard sau al casetei de dialog New Windows Messages/Event Handlers.

Puteți adăuga o funcție de tratare care să răspundă la efectuarea de către utilizator a unui clic pe controlul `Calendar`.

Adăugarea unei funcții de tratare a unui eveniment ActiveX cu ClassWizard

1. Apelați ClassWizard apăsând **Ctrl+W** sau efectuând clic pe meniul **View** și selectând **ClassWizard**.
2. Asigurați-vă că vă aflați în pagina **Message Maps**.
3. Selectați `CActiveXDlg` din caseta combinată **Class Name**, dacă nu este deja selectat.
4. Selectați identificatorul `IDC_CALENDAR` din lista **Object IDs**.
5. Efectuați un dublu clic pe mesajul **Click** din lista **Messages** pentru a apela caseta de dialog **Add Member Function**.
6. Acceptați numele implicit al funcției membru, `OnClickCalendar`, efectuând un clic pe **OK**.
7. Efectuați un clic pe butonul **Edit Code** pentru a începe editarea noii funcții de tratare.

Acum ar trebui să vedeți noua funcția de tratare `OnClickCalendar()`; această funcție va fi apelată de fiecare dată când se efectuează un clic pe controlul `Calendar` și se modifică data selectată. Ați putea să afișați noua dată pe bara de titlu a casetei de dialog, adăugând în funcția de tratare codul prezentat în listingul 9.2.

LISTINGUL 9.2 LST9_2.CPP – Adăugarea unei funcții de tratare pentru evenimentul **Click** generat de controlul ActiveX `Calendar`

```
1 void CActiveXDlg::OnClickCalendar()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Convertim valoarea de tip VARIANT furnizata de calendar
6     COleDateTime dtChosenDate(m_Calendar.GetValue()); — ❶
7
8     // ** Formataam data OLE pentru o afisa
9     // ** pe bara de titlu a casetei de dialog
10    SetWindowText(dtChosenDate.Format("You Chose %x"));
11 }
```

❶ Structura `VARIANT` este transformată de funcția constructor într-un obiect

În listingul 9.2, linia 6 folosește funcția membru `GetValue()` a obiectului `m_Calendar` pentru a determina data selectată curent în cadrul controlului `Calendar`. Dacă efectuați un dublu clic pe funcția membru `GetValue()` a clasei `CCalendar` din pagina **ClassView**, veți vedea că această funcție întoarce o valoare de tip `VARIANT`:

```
VARIANT CCalendar::GetValue()
{
    VARIANT result;
    InvokeHelper(0xc, DISPATCH_PROPERTYGET, VT_VARIANT,
        (void*)&result, NULL);
    return result;
}
```

Tipul `VARIANT` este o structură care poate reține mai multe tipuri de date diferite. Așa ceva este posibil prin existența unor membri pentru fiecare tip de date, acești membri fiind adunați într-o uniune (în C++, o uniune păstrează toți membrii la aceeași adresă pentru a economisi spațiu). Există, de asemenea, un membru `vt` de tip `VARTYPE`, a cărui valoare reprezintă tipul de date reținut curent în structură.

Veți vedea că structura `VARIANT` și clasa MFC care o încapsulează, `COleVariant`, sunt folosite destul de mult în lumea COM/OLE. Acestea permit aplicațiilor să transfere date ale căror semnificații pot fi deosebite în diferite limbaje, oferind astfel un mecanism generic pentru stocarea și extragerea acelor date. Din fericire veți opera destul de rar cu variante, pentru că multe dintre clasele MFC recunosc aceste structuri și le convertesc în forme mai prietenoase. În listingul 19.2, această conversie are loc în linia 6, în care structura `VARIANT` întoarce de apelul `GetValue()` este transmisă direct constructorului obiectului `dtChosenDate` în scopul stocării sale sub forma unui obiect `COleDateTime`.

Clase pentru manipularea reperelor temporale

MFC pune la dispoziție patru clase pentru manipularea și stocarea tuturor elementelor ce țin de date și ore. Inițial existau doar două clase, `CTime` și `CTimeSpan`, care se bazează pe sistemul `time_t` (valoare întreagă pe 4 octeți) din UNIX (se reține numărul de secunde scurse începând cu anul 1970). Numai că o precizie de ordinul secundelor și un interval de date limitat între 1970 și 2038 s-au dovedit prea restrictive pentru multe aplicații. Prin urmare, aplicațiile OLE moderne folosesc două clase noi, `COleDateTime` și `COleDateTimeSpan`.

`COleDateTime` are la bază o structură de tip `DATE` (care nu este, în realitate, decât o valoare `double`). Capacitatea acestei soluții de stocare permite clasei să acopere date între 1 ianuarie 100 și 31 decembrie 9999, cu o rezoluție de aproximativ 1 milisecundă. Diferența dintre două valori `COleDateTime` poate fi reprezentată și manipulat prin intermediul obiectului `COleDateTimeSpan`.

`COleDateTime` este un obiect care reține și manipulează date și ore. Dispune de o utilă funcție `Format()`, care poate fi utilizată pentru a transforma data stocată într-un șir `CString` elegant formatat prin intermediul unor indicatori de formatare specificați în apel. Indicatorul `%x` este utilizat în linia 10 de către funcția `Format()` pentru a genera o reprezentare a datei stocate în obiectul `COleDateTime` având forma generală: Ziua_din_Săptămână, Luna Ziua_din_Lună, Anul. (Spre exemplu, Monday, April 21, 1998.)

Data formatată într-un șir `CString` poate fi transmisă direct funcției `SetWindowText()`, iar aceasta va afișa pe bara de titlu a casetei de dialog data selectată de către utilizator.

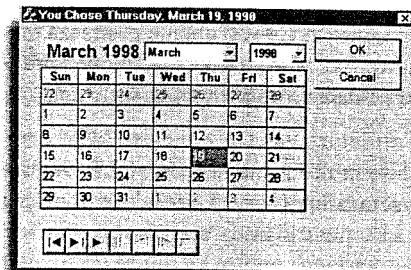
Dacă asamblați și rulați programul după adăugarea în funcția de tratare a codului prezentat, efectuarea unui clic pe calendar ar trebui să ducă la afișarea datei selectate pe bara de titlu a casetei de dialog, așa cum o ilustrează figura 9.8.

VEDEȚI...

- Pentru detalii privind tratarea mesajelor și a evenimentelor, vedeți pagina 75.
- În legătură cu crearea propriilor controale `ActiveX`, vedeți pagina 605.
- Pentru a afla mai multe despre `COM` și `OLE`, vedeți pagina 581.

FIGURA 9.8

Utilizarea evenimentelor generate de controlul `ActiveX` pentru a actualiza data afișată.

**CAPITOLUL****10****Utilizarea casetelor de dialog**

Crearea în aplicații a unor casete de dialog personalizate și a claselor corespunzătoare

Folosirea casetelor de dialog modale pentru a reține date ale aplicației

Crearea casetelor de dialog nemodale pentru cazuri deosebite

Despre utilizarea și personalizarea schimburilor și validărilor de date din casetele de dialog

Crearea unei casete de dialog

Este posibilă crearea mai multor casete de dialog, care pot fi apelate apoi din fereastra sau caseta de dialog principală a aplicației ca răspuns la interacțiunea cu utilizatorul. Pentru fiecare casetă de dialog suplimentară veți avea nevoie de o resursă de tip machetă de dialog și de o clasă asociată, derivată din `CDialog`. Macheta de dialog poate fi creată prin intermediul editorului de resurse, după care puteți utiliza `ClassWizard` pentru a crea o clasă derivată din `CDialog` care să gestioneze funcționalitatea casetei de dialog. Noua clasă poate fi dezvoltată în sensul gestionării controalelor aflate pe suprafața casetei de dialog, așa cum ați extins până acum casetele de dialog principale ale aplicațiilor de tip dialog.

Majoritatea casetelor de dialog urmează un model standard, cu care sunteți probabil familiar. Acest model constă din afișarea unei casete de dialog conținând câmpuri de opțiuni, inițializate sau nu, acordarea către utilizator a posibilității de a modifica aceste opțiuni și apoi aplicarea lor în cazul selectării unui buton **OK** sau anularea acestora la selectarea unui buton **Cancel**. Din punctul de vedere al programatorului, aceasta înseamnă, în principiu, că o casetă de dialog trebuie să conțină o copie locală a datelor care sunt supuse modificării. În cazul în care utilizatorul efectuează clic pe **OK**, va trebui să aplicați modificările asupra datelor originale ale aplicației; în caz contrar, nu aveți de făcut nimic.

Informarea utilizatorului prin intermediul `AfxMessageBox()`

Dacă aveți nevoie de o casetă de dialog doar pentru informarea utilizatorului, puteți obține o gamă largă de variante funcționale transmitând parametri corespunzători în apelul funcției `AfxMessageBox()`. Această funcție are două forme, diferența fiind dată de primul parametru. În prima formă, acest parametru este un pointer către șirul de caractere care va fi afișat, iar în a doua reprezintă identificatorul resursei de tip șir de caractere care va fi afișată. Cel de-al doilea parametru este opțional și reprezintă tipul casetei de mesaj, specificând stilul acesteia (ce butoane sunt prezente și dacă este afișată o pictogramă). Al treilea parametru vă permite să specificați un identificator pentru asistență contextuală. Funcția întoarce o valoare corespunzătoare butonului selectat.

Abordarea uzuală este implementată în general prin intermediul unei casete de dialog modale. La afișarea unei astfel de casete de dialog, restul interfeței cu utilizatorul a aplicației devine inaccesibil, obligând utilizatorul să închidă caseta de dialog înainte de a putea relua lucrul obișnuit. Aceste casete de dialog modale pot fi cascadeate astfel încât o casetă de dialog să o apeleze pe alta, determinând în acest fel transmiterea controlului către caseta de dialog cea mai recentă, aflată în prim-plan, pentru a reveni după închiderea acesteia la caseta de dialog apelantă.

Casetele de dialog nemodale reprezintă o alternativă la forma modală. Acestea sunt afișate în timp ce restul aplicației rămâne accesibil. O casetă de dialog de tip bară cu instrumente flotantă, așa cum este paleta `Controls`, reprezintă un bun exemplu de casetă de dialog nemodală.

Casetele de dialog nemodale sunt explicate pe larg mai târziu în acest capitol; oricum, ambele tipuri de casete de dialog necesită o clasă C++ care să încapsuleze funcționalitatea casetei, aceasta fiind asociată cu o resursă de tip machetă de dialog utilizată la afișarea controalelor.

VEDEȚI...

► Pentru crearea resurselor de tip machetă de dialog, vedeți pagina 54.

Adăugarea unei noi resurse de tip machetă de dialog

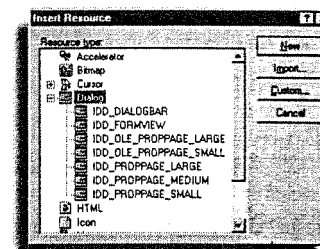
Înainte de a crea o nouă clasă pentru gestionarea unei casete de dialog, trebuie să creați o resursă de tip machetă de dialog și să adăugați în cadrul său controalele necesare.

Inserarea unei noi resurse de tip machetă de dialog

1. Selectați pagina `ResourceView` pentru a afișa resursele proiectului.
2. Efectuați un clic cu butonul drept pe elementul aflat pe primul nivel și selectați opțiunea **Insert** a meniului contextual pentru a deschide caseta de dialog `Insert Resource` (vedeți figura 10.1).

FIGURA 10.1

Caseta de dialog `Insert Resource` afișează opțiuni pentru crearea unei noi machete de dialog.



3. Selectați elementul `Dialog` de pe primul nivel pentru a insera o casetă de dialog obișnuită. Ca alternativă, puteți să optați pentru unul din tipurile de casete de dialog specializate, efectuând un clic pe simbolul plus pentru a afișa tipurile puse la dispoziție.
4. Efectuați un clic pe **New** pentru a insera noua resursă casetă de dialog. Aceasta ar trebui să fie afișată acum ca subordonată elementului `Dialog Box Resource`.
5. Efectuați un clic cu butonul drept pe caseta de dialog și selectați opțiunea **Properties** a meniului contextual pentru a afișa proprietățile machetei de dialog.
6. Acum puteți să înlocuiți identificatorul `IDD_implicit` al resursei cu un nume mai potrivit pentru caseta de dialog.

VEDEȚI...

- Pentru a afla mai multe despre crearea resurselor de tip machetă de dialog, vedeți „Crearea de machete pentru casetele de dialog”, la pagina 54.
- Pentru mai multe informații privind crearea unei aplicații de tip dialog, vedeți pagina 54.

Derivarea unei clase din `CDialog` cu ajutorul `ClassWizard`

Odată inserată noua machetă de dialog, puteți să adăugați controalele necesare și să modificați după nevoie opțiunile implicite. Pentru utilizarea acestei machete de dialog trebuie să creați o nouă clasă care să încarce și să afișeze caseta de dialog și să răspundă la

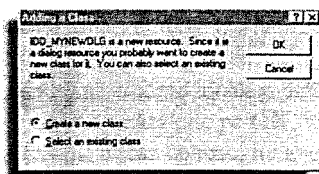
mesajele provenite de la aceasta sau de la controalele sale. Clasa de bază `CDialog` se ocupă de cea mai mare parte a încărcării și afișării unei machete de dialog. Pentru a prelua această funcționalitate, puteți să derivați propria clasă din `CDialog`. De altfel, ClassWizard automatizează acest proces în beneficiul dumneavoastră.

Crearea unei noi clase de încapsulare a unui dialog cu ajutorul ClassWizard

1. Efectuați un clic pe noua resursă machetă de dialog în cadrul secțiunii ResourceView pentru a o selecta.
2. Apăsați **Ctrl+W** sau efectuați un clic pe meniul **View** și selectați **ClassWizard** pentru a apela ClassWizard.
3. ClassWizard va sesiza că macheta de dialog este o resursă nouă și va afișa caseta de dialog Adding a Class (vedeți figura 10.2).

FIGURA 10.2

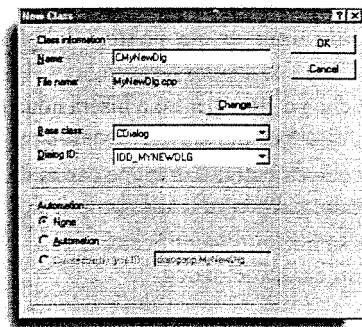
ClassWizard afișează caseta de dialog Adding a Class în momentul în care descoperă o nouă resursă machetă de dialog.



4. Efectuați un clic pe **OK** pentru a accepta opțiunea implicită de creare a unei noi clase (**Create a New Class**).
5. ClassWizard va afișa în continuare caseta de dialog New Class, ilustrată în figura 10.3.

FIGURA 10.3

Caseta de dialog New Class vă permite să stabiliți un nume pentru noua clasă de încapsulare a casetei de dialog.



6. În interiorul casetei **Name** puteți să specificați un nume pentru noua clasă de încapsulare a casetei de dialog. Prin convenție, clasele derivate din clasele MFC încep cu litera **c**.
7. Pe măsură ce tastați, numele fișierului de implementare apare în caseta **File Name**. Dacă este nevoie, puteți schimba acest nume implicit efectuând un clic pe butonul **Change**.
8. Este de preferat să acceptați clasa `CDialog` în cadrul câmpului **Base Class**, excepție făcând cazul în care implementați o pagină de proprietăți dintr-un catalog de proprietăți.

9. În caseta combinată **Dialog ID** ar trebui să fie afișat identificatorul casetei de dialog în cauză. În caz că nu sau dacă doriți să folosiți o altă machetă de dialog, puteți selecta identificatorul corespunzător din mulțimea noilor resurse de tip machetă de dialog, afișate în lista derulantă a casetei combinate.
10. Efectuați un clic pe **OK** pentru a crea noua clasă, pentru care vor fi generate fișierele de implementare (.cpp) și de definiție (.h) specificate.
11. ClassWizard va afișa apoi pagina **Message Maps** obișnuită, permițându-vă să adăugați funcții de tratare ale evenimentelor din caseta de dialog. Acum puteți crea în maniera cunoscută funcții de tratare și variabile de mapare pentru caseta de dialog și pentru controalele sale.
12. Efectuați un clic pe **OK** pentru a închide ClassWizard după ce ați încheiat adăugarea funcțiilor de tratare sau a variabilelor de mapare.

VEDEȚI...

- Pentru mai multe informații privind crearea unei aplicații de tip dialog, vedeți pagina 54.
- Pentru a afla mai multe despre tratarea mesajelor, vedeți pagina 75.

Inițializarea noii clase dialog

Odată adăugată noua clasă dialog, aceasta va fi afișată în lista de clase a proiectului din pagina ClassView. Dacă efectuați un clic pe semnul plus aflat în partea stângă a noii clase, ar trebui să puteți vedea funcția constructor (având același nume cu cel al clasei).

La efectuarea unui dublu clic pe această funcție constructor veți vedea un cod similar cu cel de aici:

```
CMYNewDlg::CMYNewDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CMYNewDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CMYNewDlg)
    // NOTE: the ClassWizard will add member initializations
    //}}AFX_DATA_INIT
}
```

Observați că un membru al clasei dialog, `CMYNewDlg::IDD`, este transmis constructorului clasei de bază, `CDialog`. În definiția clasei pe care ați creat-o ar trebui să găsiți acest membru declarat ca o valoare de tip enumerare și inițializat cu identificatorul de resursă al casetei de dialog:

```
enum { IDD = IDD_MYNEWDLG };
```

Codul generat de ClassWizard

Aveți grijă la modificarea codului generat de ClassWizard. Acesta devine foarte ușor derutat în cazul în care găsește lipsă un identificator la a cărui prezență se aștepta. Dacă adăugați propriul cod de inițializare, este de preferat să inserați noile linii după comentariile ClassWizard.

Clasa de bază `CDialog` folosește această valoare pentru a încărca resursa machetă de dialog corespunzătoare la instanțierea clasei derivate (`CMyNewDlg`). Dacă adăugați noi variabile membru prin intermediul `ClassWizard`, veți vedea, în plus, linii de inițializare pentru aceste variabile, plasate între comentariile `AFX_DATA_INIT`.

Ați putea dori să adăugați manual (adică nu cu `ClassWizard`) variabile membru în definiția clasei. Aceste variabile trebuie stabilite la valori inițiale adecvate în cadrul funcției constructor de mai sus.

La apelări ulterioare ale `ClassWizard`, veți avea posibilitatea de a selecta clasa dorită în cadrul casetei combinate **Class Name**. După adăugarea unei noi clase dialog într-o aplicație de tip dialog, în această casetă vor figura atât clasa atașată noii casete de dialog, cât și clasa casetei de dialog principale a aplicației. Atunci când adăugați noi variabile membru sau funcții de tratare a mesajelor pentru controalele dintr-o casetă de dialog, va trebui să vă asigurați că în caseta combinată **Class Name** este selectată clasa adecvată. Altfel, `ClassWizard` va efectua adăugiri în clasa selectată curent (care s-ar putea să nu fie cea pe care o vizați!).

Cataloge și pagini de proprietăți

`ClassWizard` vă permite, de asemenea, să creați o casetă de dialog paginată, cunoscută și sub numele de catalog de proprietăți. În principal, aceste casete de dialog au un comportament similar cu casetele de dialog modale, fiecare secțiune fiind o resursă machetă de dialog distinctă. Pentru a crea o casetă de dialog paginată, derivați clasele necesare din clasele `MFC CPropertySheet` și `CPropertyPage`.

Afișarea unei casete de dialog modale

Ajuns în acest punct, este momentul să vă întrebați cum va fi apelată noua casetă de dialog. De regulă, o casetă de dialog este apelată ca răspuns la o acțiune a utilizatorului, cum ar fi selectarea unui meniu sau efectuarea unui clic pe un buton aflat într-o fereastră sau casetă de dialog părinte.

Aflat în funcția de tratare corespunzătoare a ferestrei părinte (funcția care răspunde la acțiune), puteți să afișați noua casetă de dialog în doi pași simpli. Primul pas este să creați o instanță a clasei dialog. Aceasta se poate face ușor, declarând un obiect dialog local funcției de tratare, ca de pildă:

```
CMyCustomDlg dlgMyCustom(this);
```

În linia de mai sus, `CMyCustomDlg` reprezintă clasa dialog generată de `AppWizard`. Această clasă va fi astfel instanțiată la obiectul `dlgMyCustom`. Constructorului îi este transmis pointerul special `this` din C++. Pentru că funcția de tratare aparține unei clase derivate din `CWnd` (precum `CDialog`, `CView` și așa mai departe), noii casete de dialog îi va fi transmisă o informație corectă privind fereastra părinte. Vă amintiți din secțiunea anterioară că macheta de dialog va fi transmisă automat pentru inițializarea clasei de bază `CDialog`.

Pentru ca procesul de compilare să recunoască clasa dialog pe care ați derivat-o în momentul în care prelucrează codul sursă care o instanțiază (`CMyCustomDlg` din linia de

mai sus), trebuie să aveți grijă să includeți fișierul de definiție pentru această clasă (fișierul `.h`) la începutul modului care conține instanțierea, folosind în acest scop o directivă de preprocesare `#include`.

Spre exemplu, dacă definiția noii clase dialog se află în `MyCustomDlg.h`, iar în fișierul de implementare `CustomDlg.cpp` adăugați o linie de cod care instanțiază această clasă, la începutul fișierului `CustomDlg.cpp` ar trebui să inserați următoarea linie:

```
#include "MyCustomDlg.h"
```

Al doilea pas constă în afișarea și declanșarea operării modale a casetei de dialog. În acest scop este suficient să apelați funcția `DoModal()` a casetei de dialog. Ca efect, va fi afișată caseta de dialog împreună cu controalele sale componente (din macheta de dialog) și va fi iterată o buclă de mesaje locală, mesajele fiind transmise de la controalele casetei de dialog către implementarea dumneavoastră până când utilizatorul închide caseta de dialog.

În cazul în care apare o eroare la afișarea noii casete de dialog, `DoModal()` întoarce imediat un cod întreg având una din valorile `-1` sau `IDABORT`. Dacă afișarea casetei de dialog reușește, `DoModal()` revine atunci când se apelează `EndDialog()`, ceea ce închide caseta de dialog. `EndDialog()` primește o valoare întreagă ce este întoarsă apoi de `DoModal()` pe post de cod de terminare.

Implementările implicite ale funcțiilor `OnOK()` și `OnCancel()` apelează automat `EndDialog()`, transmitând codurile de terminare `IDOK` și `IDCANCEL`, întoarse apoi de către `DoModal()`. De exemplu, pentru a deschide caseta de dialog ați putea să formulați următorul apel `DoModal()`:

```
int nRetCode = dlgMyCustom.DoModal();
```

Închiderea casetelor de dialog modale cu `EndDialog()`

Dacă doriți să închideți o casetă de dialog prin cod, apelați întotdeauna la funcția `EndDialog()`, așa cum o fac `OnOK()` și `OnCancel()`, și nu folosiți `DestroyWindow()`. `EndDialog()` asigură eliberarea resurselor grafice utilizate de către caseta de dialog înainte de distrugerea acesteia. Puteți să închideți o casetă de dialog cu `EndDialog()` chiar și în interiorul funcției `OnInitDialog()`, ceea ce înseamnă că respectiva casetă va fi închisă chiar înainte de a fi afișată utilizatorului.

La închiderea casetei de dialog (de regulă prin efectuarea de către utilizator a unui clic pe **OK** sau pe **Cancel**), codul întors (de obicei, `IDOK` sau `IDCANCEL`) va fi atribuit variabilei `nRetCode`. În mod normal, acum este momentul să testați `nRetCode` pentru a efectua acțiunea corespunzătoare, în funcție de răspunsul utilizatorului, un exemplu fiind modificarea datelor aplicației dacă a fost întors `IDOK`, ca aici:

```
if (nRetCode == IDOK)
{
    // Notam modificările efectuate în dialog
}
```

Puteți, de asemenea, să închideți prin cod caseta de dialog în orice moment, apelând funcția `EndDialog()` și transmitându-i codul de terminare dorit.

Adăugarea de variabile membru pentru stocarea datelor din casetele de dialog

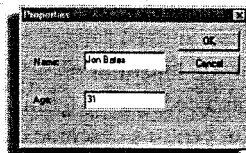
Majoritatea programelor necesită inițializarea casetelor de dialog cu o copie a datelor (sau opțiunilor) curente din cadrul aplicației. Apoi permit utilizatorului să modifice starea controalelor aflate în casetele de dialog, antrenând astfel, direct sau indirect, modificarea copiei locale din caseta de dialog a datelor aplicației. În fine, datele locale trebuie să actualizeze datele originale ale aplicației în cazul în care utilizatorul efectuează clic pe OK.

Aceasta înseamnă, în mod normal, că înainte de a apela funcția `DoModal()` a unei casete de dialog trebuie să transmiteți obiectului corespunzător datele curente ale aplicației, după care, la terminarea funcției `DoModal()` cu codul OK, trebuie să notați valorile modificate.

De multe ori puteți să păstrați copia locală din caseta de dialog a datelor aplicației în cadrul variabilelor de mapare a controalelor (cele adăugate cu ClassWizard), extrăgând valorile acestora după închiderea casetei de dialog. Spre exemplu, să presupunem că trebuie să afișați o casetă de dialog simplă care operează cu un nume și o vârstă prin intermediul a două controale de editare, așa cum se vede în figura 10.4.

FIGURA 10.4

O casetă de dialog simplă.



Ați putea să mapați peste cele două controale o variabilă `CString` (`m_strName`) care să rețină numele și o variabilă `int` (`m_nAge`) care să rețină vârsta. Aceste mapări pot fi create cu ClassWizard, având ca rezultat harta de mapare a variabilelor membru ale clasei dialog (`CMyCustomDlg`) din figura 10.5.

Puteți, mai departe, să inițializați aceste variabile, după declararea obiectului casetă de dialog, dar înainte de afișarea casetei de dialog, stabilind valorile lor direct prin cod:

```
CMyCustomDlg dlgMyCustom(this);
dlgMyCustom.m_nAge = 31;
dlgMyCustom.m_strName = "Jon Bates";
dlgMyCustom.DoModal();
```

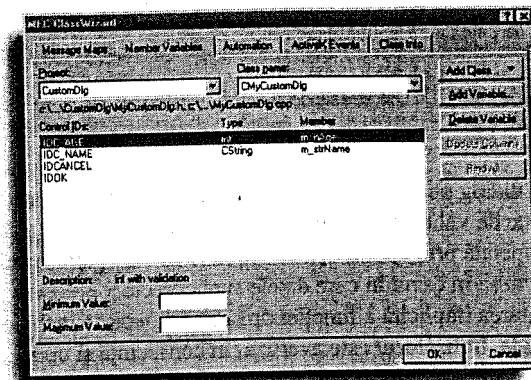
Utilizatorul poate modifica apoi aceste valori prin intermediul controalelor.

Dumneavoastră veți verifica dacă `DoModal()` a întors codul `IDOK` și veți citi valorile înscrise în variabilele membru mapate ale casetei de dialog, printr-un cod ca cel de aici:

```
if (dlgMyCustom.DoModal() == IDOK)
{
    CString strMsg;
    strMsg.Format("New Name = '%s', Age = %d",
        dlgMyCustom.m_strName, dlgMyCustom.m_nAge);
    AfxMessageBox(strMsg);
}
```

GURA 10.5

ClassWizard afișează variabilele mapate dintr-o setă de dialog simplă.



În liniile de cod de mai sus, la revenirea funcției `DoModal()`, numele modificat în caseta de editare va fi înscris în `dlgMyCustom.m_strName`, iar vârsta va fi înscrisă în `dlgMyCustom.m_nAge`. Aceste date sunt formate într-un obiect `CString` și afișate sub forma unei casete de mesaj. Într-o aplicație concretă, veți dori probabil să transferați aceste informații modificate în zona principală de date a programului.

EDEȚI...

► Pentru a vă informa în legătură cu adăugarea variabilelor membru mapate peste controale, vedeți pagina 95.

Utilizarea schimbului și validării datelor din casetele de dialog

Poate că vă întrebați cum sunt transferate valorile între variabilele membru și controalele de editare. Atunci când mapați variabile membru peste controale prin intermediul ClassWizard, este generat automat cod pentru efectuarea acestui schimb în cadrul funcției `DoDataExchange()` a casetei de dialog. Această funcție răspunde de transferul datelor în ambele direcții, înspre și dinspre controalele casetei de dialog.

Procesul de transfer se numește schimb de date și este inițiat printr-un apel al funcției `UpdateData()` a casetei de dialog. `UpdateData()` poate primi un parametru Boolean, numit `bSaveAndValidate`. Puteți transmite fie `FALSE`, pentru a actualiza controalele pe baza variabilelor membru, fie `TRUE`, pentru a înscrie valorile curente ale controalelor în variabilele membru. `UpdateData()` este apelată automat de funcția `OnInitDialog()` a clasei de bază `CDialog` pentru a efectua inițializarea controalelor la afișarea casetei de dialog. Funcția `OnOK()` a clasei de bază (apelată atunci când utilizatorul efectuează clic pe OK) este responsabilă de apelul `UpdateData()` cu parametrul `TRUE`, transferând astfel valorile din controale în cadrul variabilelor membru.

Schimbul de date nu se limitează la casetele de dialog

Procedul de utilizare a funcției `UpdateData()` în conjuncție cu `DoDataExchange()` nu se limitează doar la casetele de dialog. De fapt, aceste două funcții sunt membre ale clasei `CWnd`, ceea ce înseamnă că puteți să implementați schimb și validare de date pentru orice clasă derivată din `CWnd`.

Caseta de dialog poate să efectueze, în plus, validarea valorilor introduse de către utilizator. Aceste teste de validare sunt efectuate tot prin linii de cod adăugate în `DoDataExchange()`, care este apelată prin apelul `UpdateData()` în care parametrul `bSaveAndValidate` are valoarea `TRUE`. În cazul în care datele sunt corecte, `UpdateData()` întoarce `TRUE` (și implementarea implicită a funcției `CDialog::OnOK()` închide caseta de dialog). Dacă datele sunt invalide, utilizatorul este avertizat în consecință și `UpdateData()` întoarce `FALSE`.

Utilizarea funcțiilor pentru schimbul de date (DDX)

Puteți studia mai îndeaproape codul de validare a datelor examinând conținutul unei funcții `DoDataExchange()`. De pildă, funcția `DoDataExchange()` pentru caseta de dialog cu nume și vârstă din secțiunea anterioară ar arăta astfel:

```
void CMyCustomDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    ///{AFX_DATA_MAP(CMyCustomDlg)
    DDX_Text(pDX, IDC_AGE, m_nAge);
    DDX_Text(pDX, IDC_NAME, m_strName);
    ///}AFX_DATA_MAP
}
```

Observați că funcția primește un pointer la un obiect `CDataExchange` (`pDX`). Obiectul `CDataExchange` reține toate detaliile asociate cu direcția transferului de date și cu fereastra vizată în transfer. Pointerul `pDX` este transmis apoi, împreună cu un identificator din caseta de dialog și variabila membru asociată, în apelul funcției `DDX_Text` (generat de `ClassWizard`), în scopul implementării transferului de date. Există mai multe funcții `DDX_` care răspund de diferitele tipuri de controale și de date. Câteva dintre funcțiile mai cunoscute sunt prezentate în tabelul 10.1. `ClassWizard` adaugă automat aceste linii între comentariile `// AFX_DATA_MAP` la maparea unei variabile membru peste un control prin intermediul paginii **Member Variable**. Puteți, totodată, să adăugați manual propriile intrări, la sfârșitul funcției `DoDataExchange()`, după codul `ClassWizard`.

TABELUL 10.1 Diverse funcții DDX disponibile

Nume	Tipuri de control	Tipuri de date asociate
<code>DDX_Text</code>	Casetă de editare	<code>BYTE</code> , <code>short</code> , <code>int</code> , <code>UINT</code> , <code>long</code> , <code>DWORD</code> , <code>CString</code> , <code>LPTSTR</code> , <code>float</code> , <code>double</code> , <code>ColeCurrency</code> , <code>ColeDateTime</code>

Nume	Tipuri de control	Tipuri de date asociate
<code>DDX_Check</code>	Casetă de validare	<code>int</code>
<code>DDX_Radio</code>	Grup de butoane de opțiune	<code>int</code>
<code>DDX_LBString</code>	Casetă cu listă derulantă	<code>CString</code>
<code>DDX_LBStringExact</code>	Casetă cu listă derulantă	<code>CString</code>
<code>DDX_CBString</code>	Casetă combinată	<code>CString</code>
<code>DDX_CBStringExact</code>	Casetă combinată	<code>CString</code>
<code>DDX_LBIndex</code>	Casetă cu listă derulantă, valoarea de index	<code>int</code>
<code>DDX_CBIndex</code>	Casetă combinată, valoarea de index	<code>int</code>
<code>DDX_Scroll</code>	Bară de derulare	<code>int</code>

Aceste funcții inspectează apoi pointerul la obiectul `CDataExchange` transmis pentru a determina direcția transferului, aflată în membrul `m_bSaveAndValidate`. Puteți să testați acest indicator și să efectuați operații specifice, în funcție de direcția de transfer, printr-o linie de genul:

```
if (pDX->m_bSaveAndValidate == TRUE)
```

Funcțiile `DDX` implementează codul `WIN32` care înscrie sau extrage valorile din controale folosind mesaje `Windows` (nu uitați că fiecare control este el însuși o fereastră). Puteți să scrieți propriile funcții `DDX_` sau să adăugați cod în `DoDataExchange()` pentru a realiza interacțiuni specifice cu controalele din caseta de dialog.

Ați putea avea nevoie să apelați explicit în cod `UpdateData()` pentru a forța un schimb între controalele din caseta de dialog și variabilele membru. Așa ceva ar putea fi necesar dacă implementați funcții de tratare a mesajelor din caseta de dialog care au nevoie de valorile curente ale controalelor.

Spre exemplu, să presupunem că doriți să interceptați modificările efectuate de utilizator într-o casetă de editare dintr-un dialog pentru a afișa conținutul curent al acesteia pe bara de titlu a casetei de dialog. Ați putea să folosiți `ClassWizard` pentru a adăuga o funcție de tratare a mesajului `EN_CHANGE` generat de caseta de editare, iar pentru actualizarea titlului casetei de dialog ați putea apela funcția `SetWindowText()`.

Dar în lipsa unui apel `UpdateData(TRUE)`, variabila membru mapată nu va fi actualizată cu conținutul curent al controlului. Adăugând această linie în funcția de tratare, vă asigurați că `DoDataExchange()` este apelată și transferă conținutul controlului de editare în variabila mapată de tip `CString`.

Funcția de tratare (bazată pe numele casetei de editare din caseta de dialog cu nume și vârstă din secțiunea anterioară) ar trebui, prin urmare, să arate astfel:


```
void CMyCustomDlg::OnChangeName()
{
    UpdateData(TRUE);
    SetWindowText(m_strName);
}
```

La apăsarea unei taste va fi apelată `OnChangeName()`, iar `UpdateData(TRUE)` va actualiza variabilele membru, inclusiv `m_strName`, prin apeluri `DDX_Text`. Conținutul actualizat este afișat apoi pe bara de titlu a casetei de dialog, prin apelul `SetWindowText()`.

Puteți, de asemenea, să apelați `UpdateData(FALSE)` pentru a reseta conținutul controalelor la valorile variabilelor mapate. Așa ceva se poate dovedi necesar ca răspuns la efectuarea de către utilizator a unui clic pe un buton de anulare sau de resetare al casetei de dialog, având ca scop înscrierea în controale a valorilor inițiale.

VEDEȚI...

- Pentru a afla despre diferitele tipuri de controale, vedeți capitolul 5, „Utilizarea controalelor textuale”, capitolul 6, „Utilizarea controalelor de tip listă” și capitolul 7, „Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și dată/oră”.

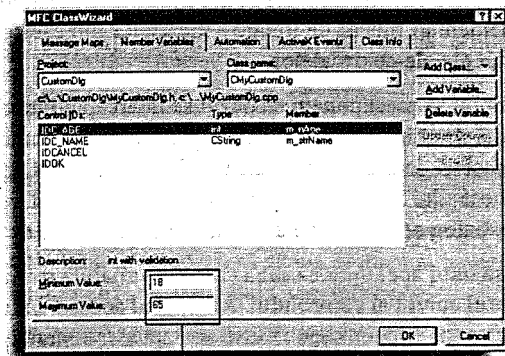
Utilizarea funcțiilor pentru validarea de date (DDV)

Atunci când adăugați anumite tipuri de variabile membru mapate într-o casetă de dialog prin intermediul ClassWizard, cum ar fi cazul unei variabile `CString` sau `int` mapată peste o casetă de editare, trebuie să specificați și valori opționale de validare. Aceste valori pot fi stabilite selectând în pagina **Member Variables** din ClassWizard una dintre noile variabilele membru mapate și modificând opțiunile de validare din partea inferioară (așa cum se vede în figura 10.6).

FIGURA 10.6

Stabilirea unui interval întreg pentru validarea unei variabile mapate.

1 Validarea



1

În cazul unui `CString` mapat peste un control de editare, puteți preciza o lungime maximă pentru a limita numărul de caractere din șirul introdus de utilizator. În cazul unui `int` mapat peste un control de editare, puteți specifica o limită inferioară și una

superioară, astfel încât utilizatorul să fie avertizat dacă întregul introdus se află în afara intervalului acceptabil.

Interceptarea excepției `CUserException` generată de eșecul validării

La eșecul unei reguli de validare este generată o excepție `CUserException`. Puteți intercepta această excepție pentru a supradefini caseta de mesaj implicită.

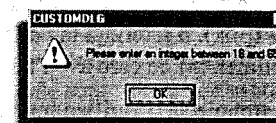
La adăugarea acestor reguli de validare, ClassWizard va genera linii DDV corespunzătoare în funcția `DoDataExchange()`. De exemplu, regula de validare ilustrată în partea inferioară a figurii 10.6 va duce la generarea următoarei linii DDV:

```
DDV_MinMaxInt(pDX, m_nAge, 18, 65);
```

Apelul funcției DDV va fi plasat imediat după apelul DDX corespunzător, astfel încât valoarea `m_nAge` să fie actualizată înainte de efectuarea validării. Dacă rezultatul validării nu satisface intervalul fixat, utilizatorul este informat printr-o casetă de mesaj corespunzătoare și apelul `UpdateData()` care a inițiat schimbul de date va întoarce valoarea `FALSE`, indicând că validarea a eșuat (vedeți figura 10.7).

FIGURA 10.7

Avertismentul implicit către utilizator în cazul eșecului unei validări numerice.



Codul generat de ClassWizard pentru validare în cazul limitării numărului de caractere introduse într-o casetă de editare ar arăta astfel:

```
DDV_MaxChars(pDX, m_strName, 15);
```

Funcțiile de validare DDV disponibile sunt prezentate în tabelul 10.2. Acestea pot fi utilizate cu orice mapare de control care permite aplicarea tipului de date corespunzător, printre aceste controale numărându-se controalele de editare, barele de derulare și grupurile de butoane de opțiune. În general, ele limitează valoarea numerică permisă între anumite limite. Puteți adăuga manual aceste apeluri de funcții DDV, dar aveți grijă să plasați noile intrări după secțiunea `AFX_DATA_MAP` rezervată pentru ClassWizard; altfel, s-ar putea să încurcați ClassWizard.

TABELUL 10.2 Funcții DDV standard generate de ClassWizard

Funcția DDV	Tipul de date aplicabil	Descriere
<code>DDV_MaxChars</code>	<code>CString</code>	Limitează numărul de caractere introduse
<code>DDV_MinMaxByte</code>	<code>BYTE</code>	Limitează numărul la un interval specificat
<code>DDV_MinMaxDouble</code>	<code>double</code>	Limitează numărul la un interval specificat

(continuare)

TABELUL 10.2 Continuare

Funcția DDV	Tipul de date aplicabil	Descriere
DDV_MinMaxDWord	DWord	Limitează numărul la un interval specificat
DDV_MinMaxFloat	float	Limitează numărul la un interval specificat
DDV_MinMaxInt	int	Limitează numărul la un interval specificat
DDV_MinMaxLong	long	Limitează numărul la un interval specificat
DDV_MinMaxUnsign	unsigned	Limitează numărul la un interval specificat
ed		

VEDEȚI...

- Pentru a afla despre diferitele tipuri de controale, vedeți capitolul 5, „Utilizarea controalelor textuale”, capitolul 6, „Utilizarea controalelor de tip listă” și capitolul 7, „Utilizarea controalelor indicator de evoluție, bară de derulare, glisor și datăloră”

Crearea propriilor funcții de validare

Puteți crea propriile funcții de validare, care să efectueze verificări mai deosebite, scriind o funcție de tip DDV_. În acest scop trebuie să creați o funcție care primește un pointer la un obiect CDataExchange (pDX), o referință la variabila mapată care va fi testată și orice alți parametri specifici suplimentari de care aveți nevoie.

Prima condiție pe care trebuie să o verifice funcția dumneavoastră de validare este dacă obiectul CDataExchange se află în modul salvează-și-validează, iar acest lucru se face verificând dacă `pDX->m_bSaveAndValidate` este TRUE. Dacă acest indicator nu este TRUE, atunci se efectuează inițializarea controalelor pe baza variabilelor membru, așa că nu sunt necesare teste de validare și funcția ar trebui să-și încheie execuția. Altfel, trebuie să validați variabila membru mapată în funcție de propriile criterii.

Dacă variabila mapată este validă, funcția se poate încheia normal; în caz contrar, trebuie să afișați o casetă de mesaj de avertizare și să apelați funcția `Fail()` a pointerului pDX la obiectul CDataExchange pentru a informa funcția `UpdateData()` apelantă că validarea a eșuat.

De exemplu, dacă ați dori să rafinați testul de vârstă din exemplul precedent pentru a exclude persoanele de 31 de ani, ați putea să scrieți o funcție de validare ca aceasta:

```
void DDV_ValidateAge(CDataExchange* pDX, int& nCheckAge)
{
    if (pDX->m_bSaveAndValidate &&
        (nCheckAge < 18 || nCheckAge > 65 || nCheckAge == 31))
    {
        AfxMessageBox(
            "Ages must be between 18 and 65, but not 31!");
        pDX->Fail();
    }
}
```

Deoarece parametrul `nCheckAge` este o referință la variabila mapată, aveți posibilitatea de a reseta valoarea acesteia la o valoare validă implicită în cazul în care validarea sa eșuează.

Așadar, funcția `DDV_ValidateAge()` poate fi apelată în cadrul funcției `DoDataExchange()`, după ce variabila membru a fost actualizată de către funcția `DDX` corespunzătoare printr-un apel de forma:

```
DDV_ValidateAge(pDX, m_nAge);
```

Validarea datelor din dialoguri

În cazul în care adăugați un apel `DDV_...` în funcția supradefinită `DoDataExchange()`, acesta ar trebui să urmeze imediat după apelul `DDX_...` corespunzător membrului de date pe care-l validați. Pentru o listă completă a macrodefinițiilor pentru schimb de date care sunt disponibile în MFC, vedeți fișierul `afxdd.h`.

Validarea pe care ați implementat-o ar trebui acum să accepte vârste între 18 și 65 de ani, mai puțin vârsta de 31 de ani.

Utilizarea casetelor de dialog nemodale

O casetă de dialog nemodală nu acaparează interfața unei aplicații, așa cum o face o casetă de dialog modală. În schimb, o casetă de dialog nemodală este afișată de obicei pentru a completa funcționalitatea tipică a aplicației prin transmiterea de mesaje și de detalii către aplicația apelantă. Casetele de dialog nemodale sunt folosite de regulă ca bare cu instrumente flotante, permițând utilizatorului să efectueze clic pe butoanele casetei de dialog pentru a indica opțiuni privind modul de lucru sau alegeri exprimate.

O casetă de dialog nemodală folosește o machetă de dialog obișnuită pentru a specifica dispunerea controalelor. Cu toate acestea, în mod normal veți înlătura butoanele OK și Cancel, specifice operării modale și devenite irelevante.

Se folosește, de asemenea, o clasă de încapsulare derivată din `CDialog`, care poate fi creată pe baza machetei prin intermediul `ClassWizard`. Diferențele dintre casetele de dialog modale și cele nemodale se află în modul de apelare și închidere a casetei de dialog.

Crearea și distrugerea casetelor de dialog nemodale

În loc să intrați într-o buclă modală prin apelul `DoModal()` de afișare a unei casete de dialog, ați putea să apelați o casetă de dialog nemodală printr-un apel al funcției `Create()` a clasei `CDialog`. O posibilitate este să plasați apelul `Create()` în constructorul clasei asociate casetei de dialog, astfel încât să creați o instanță odată cu instanțierea unui obiect de tipul clasei. Ca alternativă, ați putea dori să separați instanțierea clasei și apelul `Create()` pentru a permite un cod intermediar de inițializare. Funcția `Create()` primește doi parametri; primul reprezintă identificatorul machetei dialogului care se creează, iar al doilea este un pointer opțional către un obiect reprezentând fereastra părinte (în mod implicit, fereastra principală a aplicației). `Create()` indică reușita, întorcând valoarea TRUE, sau eșecul, întorcând valoarea FALSE.

În momentul în care `Create()` revine indicând succesul, fereastra casetei de dialog modale există, dar nu este vizibilă. Pentru a o face vizibilă este nevoie de un apel al funcției `ShowWindow()` a ferestrei casetei de dialog, transmițând valoarea simbolică `SW_SHOW` pentru afișare.

Dacă o casetă de dialog modală poate fi încapsulată fără probleme în interiorul unei funcții de tratare, casetele de dialog nemodale vor trebui accesate din alte puncte ale programului. Din această cauză, o casetă de dialog nemodală este bine să fie creată dinamic, prin intermediul operatorului `new` din C++, reținând adresa ei într-o variabilă pointer globală sau membru.

Caseta de dialog poate fi închisă la nevoie, distrugând-o prin intermediul operatorului `delete` din C++. Destructorul clasei de bază `CDialog` se va ocupa de închiderea ferestrei casetei de dialog.

Puteți experimenta aceste tehnici creând o aplicație de tip dialog cu AppWizard. Inserați o nouă resursă de tip machetă de dialog pentru caseta de dialog nemodală, stabilind identificatorul acesteia la `IDC_MODELESS` (această operație a fost descrisă mai devreme, în secvența de pași „Inserarea unei noi resurse de tip machetă de dialog”). Creați o clasă de încapsulare a casetei de dialog și numiți-o `CModeless`, ghidându-vă după secvența de pași „Crearea unei noi clase de încapsulare a unui dialog cu ajutorul ClassWizard”, prezentată anterior.

Odată adăugată noua clasă, `CModeless` ar trebui să figureze în lista de clase din pagina ClassView a secțiunii spațiului de lucru al proiectului. Dacă efectuați un clic pe semnul plus din dreptul clasei pentru a afișa funcțiile sale membru, veți vedea funcția constructor `CModeless()`. Efectuați un dublu clic pe această funcție constructor pentru a o putea edita. Acum puteți să inserați liniile de cod cu apelurile `Create()` și `ShowWindow()`, ca în listingul 10.1.

LISTINGUL 10.1 LST10_1.CPP – Adăugarea apelurilor `Create()` și `ShowWindow()` în constructorul casetei de dialog nemodale

```
1 CModeless::CModeless(CWnd* pParent /*=NULL*/)
2     : CDialog(CModeless::IDD, pParent)
3 {
4     //{{AFX_DATA_INIT(CModeless)
5     // NOTE: the ClassWizard will add member initializa
6     //{{AFX_DATA_INIT
7
8     // ** Cream fereastra casetei de dialog nemodale
9     if (Create(CModeless::ID, pParent)) — ❶
10    {
11        // ** Crearea a reusit, asa ca afisam fereastra
12        ShowWindow(SW_SHOW); — ❷
13    }
14 }
```

❶ Caseta de dialog nemodală este creată prin apelul funcției `Create()`.

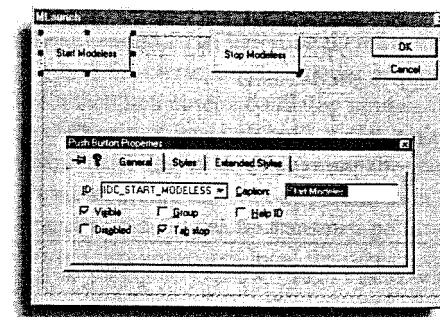
❷ Caseta de dialog nemodală creată este afișată de funcția `ShowWindow()`.

Liniile adiționale (de la 8 la 13) din listingul 10.1 stabilesc operarea nemodală a casetei de dialog. Linia 9 apelează funcția `Create()`, transmițând enumeratorul `CModeless::IDD` generat de ClassWizard, care conține identificatorul machetei de dialog asociate, și pointerul `pParent` la fereastra părinte. Dacă funcția `Create()` întoarce valoarea `TRUE` care indică reușita apelului, în linia 12 este apelată `ShowWindow()`, transmițând valoarea simbolică `SW_SHOW` pentru a afișa noua casetă de dialog.

Mai departe, ați putea să adăugați în caseta de dialog principală un buton având identificatorul `IDC_START_MODELESS`, pe care să-l folosiți pentru apelarea casetei de dialog nemodale, ca în figura 10.8, și un alt buton, având identificatorul `IDC_STOP_MODELESS`, care să o închidă.

FIGURA 10.8

Adăugarea în caseta de dialog principală a aplicației a butoanelor pentru deschiderea și închiderea casetei de dialog nemodală.



Pentru a efectua deschiderea și închiderea noii casete de dialog nemodale, puteți să creați funcții de tratare a mesajelor `BN_CLICKED` provenite de la cele două butoane din caseta de dialog principală. Apoi veți adăuga codul de implementare a acestor funcții de tratare, cod care va fi responsabil de deschiderea și închiderea casetei de dialog nemodale (vedeți listingul 10.2). Funcția de tratare `OnStartModeless()` va apela o instanță a casetei de dialog nemodale la efectuarea unui clic pe butonul corespunzător, mai puțin în cazul în care caseta este deja deschisă. Funcția de tratare `OnStopModeless()` va distruge caseta de dialog nemodală dacă aceasta este deschisă.

LISTINGUL 10.2 LST10_2.CPP – Crearea și distrugerea casetei de dialog nemodale prin intermediul funcțiilor care răspund la efectuarea de clic asupra butoanelor din caseta de dialog principală

```
1 // ** includem definitia clasei folosite
2 #include "modeless.h"
3
4 // ** Declaram un pointer global la caseta de dialog nemodala
5 CModeless* g_pDlgModelss = NULL; — ❶
6
7 void CMLaunchDlg::OnStartModeless()
```

(continuare)

LISTINGUL 10.2 Continuare

```

8 {
9     // ** Cream un nou dialog in cazul in care pointerul
10    // ** global are valoarea NULL, ceea ce arata ca
11    // ** dialogul nu este curent deschis
12    if (!g_pDlgModeless) ————— ❷
13        g_pDlgModeless = new CModeless(this);
14 }
15
16 void CMLaunchDlg::OnStopModeless()
17 {
18     // ** Daca pointerul la dialogul nemodal este valid,
19     // ** dialogul poate fi inchis si distrus
20     if (g_pDlgModeless)
21     {
22         delete g_pDlgModeless; ————— ❸
23         g_pDlgModeless = NULL;
24     }
25 }

```

❶ Pointerul global va putea fi accesat de oriunde din aplicație, dar ar putea necesita o declarație extern.

❷ În cazul în care caseta de dialog nu este deschisă, ea poate fi creată generând pur și simplu un nou obiect CModeless.

❸ Destructorul va închide automat fereastra casetei de dialog.

Linia 2 din listingul 10.2 are ca efect includerea definiției de clasă necesare pentru CModeless, aflată în fișierul modeless.h. În linia 5 este declarat un pointer global la o casetă de dialog CModeless (g_pDlgModeless), fiind inițializat cu NULL. Funcția OnStartModeless() verifică dacă pointerul g_pDlgModeless este NULL, asigurându-se că nu este deja deschisă o casetă de dialog înainte de a crea una nouă prin apelul din linia 13, în care este transmisă caseta de dialog principală ca fereastră părinte, prin intermediul pointerului this. Caseta de dialog va fi afișată imediat după creare, așa cum sunt implementate apelurile Create() și ShowWindow() în funcția constructor a casetei de dialog.

Funcția OnStopModeless() din linia 16 este apelată la efectuarea unui clic pe butonul **Stop Modeless**. Linia 20 testează pointerul g_pDlgModeless pentru a asigura existența casetei de dialog (dacă pointerul nu are valoarea NULL). Caseta de dialog este apoi distrusă în linia 22 (închizând fereastra acesteia în cadrul destructorului). În fine, pointerul la caseta de dialog capătă valoarea NULL în linia 23, astfel că efectuarea unui clic pe butonul **Start Modeless** va putea crea o nouă instanță.

Dacă asamblați și rulați aplicația după efectuarea acestor modificări, veți putea să creați și să închideți caseta de dialog dinamic, efectuând clic pe cele două butoane.

VEDEȚI...

➤ Pentru a afla despre crearea funcțiilor de răspuns la mesajele/evenimentele generate de clicurile asupra butoanelor, vedeți pagina 74.

Înscrierea și citirea datelor dintr-o casetă de dialog nemodală

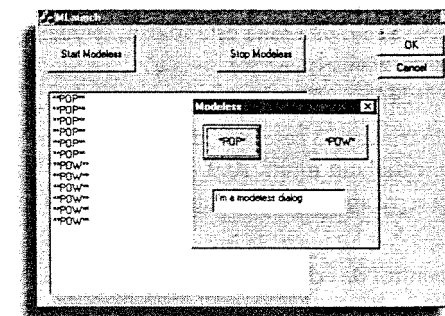
Fixarea valorilor sau apelurile funcțiilor membru ale unei casete de dialog nemodale se pot efectua pe toată durata de viață a acesteia, prin intermediul pointerului de acces corespunzător. La transferul de date între controale și variabilele membru mapate, prin UpdateData() și DoDataExchange(), se aplică regulile deja cunoscute. Puteți să fixați valorile variabilelor membru în cadrul aplicației, apelând apoi UpdateData() cu parametrul FALSE pentru a transfera în controale conținutul variabilelor membru.

Este posibil, de asemenea, să apelați funcții sau să fixați variabile membru ale altor obiecte ale aplicației în cadrul casetei de dialog modale, ca răspuns la interacțiunea utilizatorului cu controalele. În acest scop, trebuie să transmiteți casetei de dialog modale pointeri la obiectele respective ale aplicației (sau să folosiți pointeri globali), astfel încât să fie posibilă accesarea acelor obiecte. Constructorul casetei de dialog modale este un loc bun pentru transmiterea acestor pointeri. Puteți să modificați lista de parametri ai constructorului astfel încât la crearea casetei de dialog să fie transmiși pointeri la obiectele respective ale aplicației. Acești pointeri pot fi reținuți apoi local de către caseta de dialog modală, sub formă de variabile membru, astfel că oricare din funcțiile componente va putea accesa obiectele în cauză. Nu uitați să adăugați directivele de preprocesare #include necesare la începutul fișierului de definiție al clasei casetei de dialog nemodale, astfel încât compilatorul să recunoască declarațiile noilor variabile membru de tip clasă.

Pentru a ilustra cele de mai sus puteți să dezvoltați exemplul de aplicație. De pildă, puteți să adăugați un control listă în caseta de dialog principală și să trimiteți mesaje prin intermediul a două noi butoane plasate pe caseta de dialog nemodală. Efectuarea acestor modificări ar duce la o aplicație asemănătoare celei înfățișate în figura 10.9.

FIGURA 10.9

Interacțiunea dintre o aplicație și o casetă de dialog nemodală a sa.



Mai întâi, trebuie modificat constructorul casetei de dialog nemodale pentru a asigura că pointerul către părinte transmis este CMLaunchDlg, și nu pointerul CWnd curent. Aceasta se poate face ajustând ca mai jos definiția constructorului din cadrul fișierului de antet care

conține definiția clasei `CModeless` (`Modeless.h`), astfel încât singurul parametru acceptat să fie un pointer la `CMLaunchDlg`:

```
CModeless(CMLaunchDlg* pParent); // standard constructor
```

Închiderea casetelor de dialog nemodale cu `DestroyWindow()`

Dacă o casetă de dialog nemodală conține butoanele **OK** sau **Cancel**, trebuie să supradefiniți implementările implicite `OnOK()` și `OnCancel()`, deoarece acestea vor apela `EndDialog()`, care ascunde caseta de dialog fără a o distruge. Pentru a închide o casetă de dialog nemodală va trebui să apelați `DestroyWindow()`, ceea ce se poate face în noile implementări `OnOK()` și `OnCancel()`.

Editarea acestui fișier se face efectuând un dublu clic pe clasa `CModeless` din lista cu clasele proiectului, listă afișată în pagina `ClassView` a secțiunii spațiului de lucru al proiectului.

La începutul fișierului cu definiția clasei `CModeless` va trebui să adăugați directiva de preprocesare `#include` de mai jos, astfel încât compilatorul să recunoască clasa `CMLaunchDlg`:

```
#include "MLaunchDlg.h"
```

Pentru a reține pointerul transmis trebuie, de asemenea, să adăugați o variabilă membru corespunzătoare de tip pointer; aceasta poate fi declarată ca membru protejat, după operatorul `protected`, ca aici:

```
CMLaunchDlg* m_pParent;
```

Pentru ca pointerul transmis să fie reținut în acest pointer membru, veți modifica funcția constructor în sensul inițializării pointerului membru cu parametru `pParent` primit, astfel:

```
CModeless::CModeless(CMLaunchDlg* pParent)
    : CDialog(CModeless::IDD, pParent), m_pParent(pParent)
```

Observați că `pParent`, caseta de dialog părinte, poate fi transmisă ca fereastră părinte clasei de bază `CDialog`. `pParent` este folosit, totodată, pentru a inițializa membrul local prin secvența `m_pParent(pParent)`.

În continuare, puteți să adăugați în caseta de dialog principală o casetă cu listă care să afișeze mesajele trimise de caseta de dialog nemodală, așa cum se vede în caseta de dialog principală a aplicației din figura 10.9. Folosiți `ClassWizard` pentru a mapa peste această listă o variabilă `m_DisplayList` de tip `CListBox`, ca în figura 10.10.

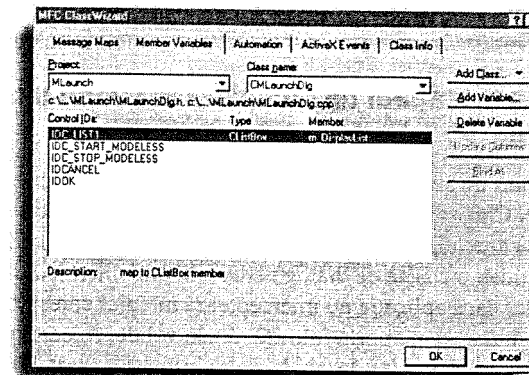
Mai departe, puteți să adăugați în resursa machetă a casetei de dialog nemodale două butoane și un control de editare. Butoanele vor fi utilizate pentru a crea două funcții de tratare care vor trimite două mesaje distincte către noua casetă cu listă din caseta de dialog principală a aplicației. Apelați la `ClassWizard` pentru crearea funcțiilor de tratare a mesajelor `BN_CLICKED` provenite de la cele două butoane, după care implementați în cadrul lor adăugarea unui șir în caseta cu listă din fereastra principală, prin intermediul codului de pe pagina următoare:

```
void CModeless::OnPop()
{
    m_pParent->m_DisplayList.AddString("***POP***");
}

void CModeless::OnPow()
{
    m_pParent->m_DisplayList.AddString("***POW***");
}
```

FIGURA 10.10

Maparea noii casete cu listă din caseta de dialog principală a aplicației.



Observați că noile funcții de tratare folosesc noul pointer `CMLaunchDlg` membru, `m_pParent`, pentru a accesa variabila `m_DisplayList` din caseta de dialog principală a aplicației. Funcția `AddString()` oferită de casetă cu listă adaugă în listă șirul specificat, indicând că utilizatorul a efectuat clic pe unul dintre butoanele casetei de dialog nemodale.

Pentru a ilustra accesarea casetei de dialog nemodale din programul principal, puteți să adăugați un control casetă de editare pe suprafața machetei acesteia apelând la editorul de resurse. Folosiți `ClassWizard` pentru a mapa peste controlul de editare o variabilă `Cstring` numită `m_DispmMsg`.

Acest membru mapat poate fi accesat acum de oriunde din program atât timp cât caseta de dialog nemodală este deschisă, folosind pointerul `g_pDlgModeless`. Pentru a înscrie un șir de caractere în cadrul controlului va trebui să forțați un schimb de date, apelând `UpdateData()` și transmitând indicatorul `FALSE`. Toate acestea sunt implementate prin următoarele linii de cod:

```
if (g_pDlgModeless) // Ne asigurăm ca dialogul este deschis
{
    g_pDlgModeless->m_DispmMsg=CString("I'm a modeless dialog");
    g_pDlgModeless->UpdateData(FALSE);
}
```

Când se folosește o casetă de dialog modală la nivel de sistem

Una dintre opțiunile de stil pentru o machetă de dialog vă permite să creați o casetă de dialog modală la nivel de sistem. Aceasta va împiedica utilizatorul să acceseze o altă fereastră sau aplicație cât timp caseta de dialog este deschisă. Utilizați casete de dialog modale la nivel de sistem numai pentru a avertiza utilizatorul în acele situații în care operații efectuate de alte aplicații ar putea altera configurația sistemului de operare.

VEDEȚI...

- Pentru a afla despre crearea funcțiilor de răspuns la mesajele/evenimentele generate de clicurile asupra butoanelor, vedeți pagina 74.
- Pentru mai multe amănunte privind utilizarea controalelor de tip listă, vedeți pagina 114.

Tratarea mesajului de închidere în cadrul unei casete de dialog nemodale

A mai rămas un singur aspect de abordat în ceea ce privește utilizarea casetelor de dialog nemodale. Chiar dacă ați înlăturat butoanele OK și Cancel specifice operării modale, butonul cu cruciuliță pentru închiderea ferestrei, aflat în colțul din dreapta sus, a rămas. Dacă închideți o casetă de dialog nemodală prin intermediul acestui buton, fereastra va fi închisă, dar dacă aplicația nu interceptează mesajul corespunzător, pointerul global la caseta de dialog și memoria folosită rămân valide. Aceasta poate duce la pierderi de memorie și nu veți putea deschide o altă instanță a casetei de dialog nemodale până când nu este închisă prima instanță.

Rezolvarea acestei probleme se face interceptând mesajul WM_CLOSE în cadrul casetei de dialog nemodale pentru a elibera memoria și a reseta pointerul. Prin intermediul casetei de dialog New Windows Message and Event Handlers puteți să adăugați o funcție de tratare care să fie apelată în momentul în care utilizatorul efectuează un clic pe butonul de închidere. Deschideți această casetă de dialog efectuând un clic cu butonul drept pe clasa CModelless din pagina ClassView și selectând apoi opțiunea Add Windows Message Handler.

Liniile de cod următoare ilustrează modificările care trebuie aduse funcției de tratare OnClose() din exemplul nostru pentru a șterge obiectul casetă de dialog și a reseta pointerul global:

```
extern CModelless* g_pDlgModelless;
void CModelless::OnClose()
{
    CDialog::OnClose();
    delete this;
    g_pDlgModelless = NULL;
}
```

Remarcați că declarația pointerului global este de tip extern; acest specificator asigură că este utilizat același pointer g_pDlgModelless care a fost declarat în modulul MLaunchDlg.cpp. Funcția de tratare apelează apoi funcția OnClose() a clasei de bază pentru a efectua prelucrările obișnuite. După aceea este eliberată memoria ocupată de

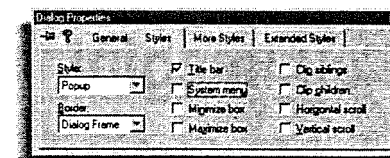
obiectul casetă de dialog nemodală (this). În fine, pointerul g_pDlgModelless este resetat la NULL pentru a permite deschiderea unei noi instanțe a casetei de dialog nemodale de către caseta de dialog principală a aplicației.

Înlăturarea butonului de închidere

În loc să interceptați mesajul WM_CLOSE, puteți să împiedicați utilizatorul să închidă caseta de dialog deselectând opțiunea **System Menu** dintre proprietățile asociate machetei de dialog în editorul de resurse (vedeți figura 10.11). În acest fel vor fi înlăturate butonul de închidere și meniul ferestrei, iar utilizatorul nu va mai avea posibilitatea de a închide caseta de dialog. Așa ceva ar putea fi de dorit dacă trebuie să mențineți deschisă caseta de dialog nemodală.

FIGURA 10.11

Înlăturarea meniului de sistem al casetelor de dialog nemodale prin intermediul editorului de resurse.



Lucrul cu imagini, bitmap-uri și pictograme

Despre crearea, importarea și editarea pictogramelor
și a imaginilor bitmap

Afișarea de imagini în casetele de dialog

Implementarea unor butoane grafice

Folosirea imaginilor în cadrul controalelor listă, arbore
și casetă combinată

Imaginile grafice domină mediul Windows. Lucrurile stau astfel din mai multe motive. Imaginile au caracter internațional, ocupă mai puțin spațiu decât echivalentele textuale și arată bine. Principalul scop al utilizării imaginilor grafice este să ajute utilizatorul să recunoască un program sau o funcție anume mai rapid decât prin parcurgerea unui text descriptiv.

Aplicațiile Windows folosesc mai multe tipuri de imagini grafice.

- O imagine *pictogramă* este asociată cu însăși aplicația și este afișată în Windows Explorer. De asemenea, pictograma unei aplicații este afișată de regulă pe suprafața de lucru pentru a indica o scurtătură.
- Imaginile *bitmap* sunt folosite în cadrul ecranelor introductive și pe butoanele de pe barele cu instrumente, putând fi plasate și în casetele de dialog sau în alte ferestre.
- Imaginile *cursor* sunt folosite pentru a modifica reprezentarea grafică a cursorului de mouse. Acestea se întâlnesc frecvent în programele de desenare, având rolul de a sugera cum poate fi editat sau deplasat un obiect selectat de pe ecran.

Editorul de resurse din Developer Studio vă permite să creați și să editați fiecare tip de imagine. Puteți să adăugați într-un proiect câte pictograme, bitmap-uri și cursoare doriți. Mai departe este responsabilitatea dumneavoastră de a implementa codul care să folosească resursele imagine când și cum considerați de cuviință.

Utilizarea editorului de imagini

Editorul de resurse de tip imagine vă permite să creați oricare dintre tipurile de imagine (bitmap, cursor, pictogramă și bară cu instrumente). Puteți să desenați cu mâna liberă sau să folosiți diferitele opțiuni și instrumente care vă ajută la trasarea de figuri pline și contururi. Fereastra editorului este în mod normal împărțită în două secțiuni, în stânga fiind afișată imaginea în mărime naturală, iar în dreapta fiind afișată o versiune mărită. Aceasta din urmă permite distingerea fiecărui pixel în parte în scopul unei editări cât mai precise. Ambele imagini sunt actualizate automat în timpul editării. Indiferent de tipul resursei editate, multe dintre funcțiile de editare grafică sunt identice.

Meniul **Image** este disponibil oricând este activată o fereastră de editare a unei imagini. Acest meniul conține mai multe instrucțiuni de desenare. Opțiunea **Invert Colors** oferă posibilitatea de a inversa culorile dintr-o zonă selectată a unui bitmap. În plus, o zonă selectată poate fi oglindită sau rotită. Tot prin intermediul meniului **Image** puteți să definiți și să salvați propriile palete de culori.

Instrumentele de desenare se află în caseta cu instrumente Graphics, iar culorile se selectează din paleta Colors, așa cum se vede în figura 11.1. În cazul în care cele două casete cu instrumente nu sunt afișate, urmați pașii de mai jos.

Afișarea casetelor cu instrumente Graphics și Colors

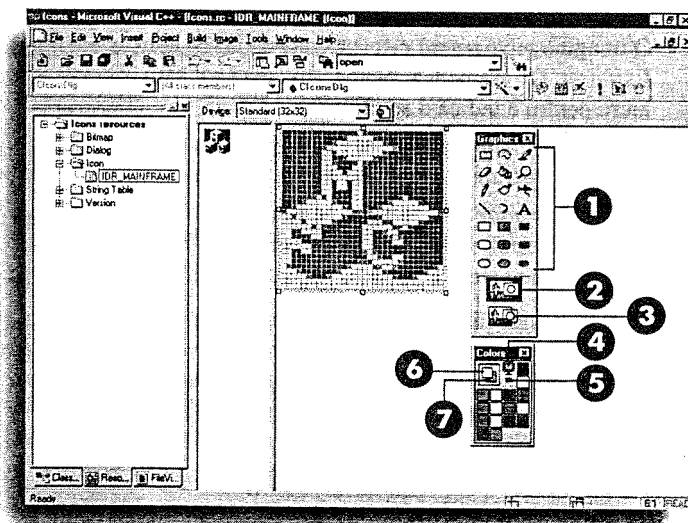
1. Selectați **Customize** din meniul **Tools**. Va fi afișată caseta de dialog **Customize**.
2. Selectați pagina **Toolbars**.
3. În cadrul listei **Toolbars**, validați elementele **Graphics** și **Colors**.

4. Efectuați un clic pe **Close**.
5. Ca alternativă, plasați mouse-ul pe bara de meniu din partea superioară a ecranului și efectuați un clic cu butonul drept.
6. Selectați **Graphics** sau **Colors** din meniul contextual afișat.

FIGURA 11.1

Editarea unei pictograme.

- 1 Instrumente de editare și desenare
- 2 Desenare opacă
- 3 Desenare transparentă
- 4 Culoarea ecranului
- 5 Culoarea inversă
- 6 Culoarea de desenare
- 7 Culoarea de fundal



Caseta cu instrumente Graphics oferă toate facilitățile tipice care se întâlnesc în aplicațiile de editare a imaginilor bitmap. Spre exemplu, există un spray, un selector de culoare, selectori de zone rectangulare și poligonale, contururi și figuri pline etc. În partea inferioară a casetei Graphics se află un selector de opțiuni care se utilizează pentru a alege stiluri de pensule, grosimi de linie și alte elemente specifice instrumentului selectat. La editarea imaginilor, rețineți că puteți utiliza opțiunile **Undo** și **Redo** din meniul **Edit** sau combinațiile de taste **Ctrl+Z** și, respectiv, **Ctrl+Y**.

Paleta Colors vă permite să selectați culoarea de desenare sau de fundal, efectuând clic cu butonul stâng sau, respectiv, drept al mouse-ului. Instrumentul de desenare selectat curent folosește culoarea de desenare atunci când este apăsat butonul stâng al mouse-ului și culoarea de fundal atunci când este apăsat butonul drept al mouse-ului.

În ceea ce privește imaginile cursor și pictogramă, paleta Colors oferă două elemente suplimentare, și anume culoarea ecranului și culoarea inversă (vedeți figura 11.1). Pixelii desenați cu culoarea ecranului vor fi transparenti la afișarea imaginii, aceasta căpătând astfel un aspect neregulat. Pixelii desenați cu culoarea inversă vor fi afișați în negativ atunci când pictograma este deplasată.

Editorul de imagini permite, de asemenea, operații de decupare și lipire în și din Clipboard. De exemplu, dacă doriți mai multe imagini similare, puteți să selectați o porțiune dintr-o imagine și să o lipiți în alta. La lipire puteți opta pentru modul opac sau transparent,

selectând opțiunea corespunzătoare din caseta cu instrumente sau selectând opțiunea **Draw Opaque** din meniul **Image**.

VEDEȚI...

► Pentru detalii legate de editarea resurselor de tip bară cu instrumente, vedeți pagina 300.

Crearea și editarea resurselor de tip pictogramă

Din punct de vedere vizual, distincția între pictograme și bitmap-uri este greu de făcut, dar există o serie de deosebiri importante. Un bitmap este o secvență de date care reprezintă o imagine în culori formată din pixeli. O pictogramă se compune din două bitmap-uri; primul reprezintă o imagine în culori, iar cel de-al doilea este un bitmap mască. Îmbinarea informațiilor conținute în aceste două bitmap-uri face posibilă existența zonelor de transparentă și de inversare în cadrul pictogramelor.

Veți întâlni de multe ori pictograme care au aspecte neregulate – rotunde, de pildă. În realitate, toate pictogramele sunt dreptunghiulare; masca de transparentă este cea care le permite să pară neregulate. O pictogramă poate conține mai multe imagini care diferă prin dimensiune și număr de culori, fiind destinate unor dispozitive de afișare diferite.

Modificarea pictogramei implicite MFC

Aplicațiile generate de AppWizard conțin întotdeauna o pictogramă implicită. Aceasta reprezintă MFC sub o formă tridimensională. Identificatorul pictogramei este `IDR_MAINFRAME`, iar pictograma este cea asociată aplicației și afișată în caseta de dialog **About**. Pentru a personaliza pictograma aplicației din cadrul unui proiect, urmați secvența de pași „Modificarea pictogramei implicite MFC”.

De regulă, pictogramele asociate cu o aplicație sau cu un tip anume de document conțin două imagini cu 16 culori: standard (32x32 pixeli) și de dimensiune redusă (16x16 pixeli). Aceste imagini ale pictogramei sunt înregistrate de Windows pentru a putea fi afișate în Windows Explorer, în meniul de start sau pe bara de sarcini.

Modificarea pictogramei implicite MFC

1. În cadrul paginii ResourceView, expandați catalogul **Icon** și efectuați un dublu clic pe `IDR_MAINFRAME`. Pictograma implicită MFC este afișată în editorul de resurse (vedeți figura 11.1).
2. Editați imaginea **Standard (32x32)** a pictogramei, folosind casetele cu instrumente Colors și Graphics. Despre editarea unei imagini am discutat în secțiunea „Utilizarea editorului de imagini”, mai devreme în acest capitol.
3. În cadrul casetei combinate **Device**, selectați și editați imaginea **Small (16x16)** a pictogramei. În cazul în care nu există o versiune redusă a pictogramei, Windows va crea una pornind de la versiunea standard, folosind-o apoi pentru afișarea în Windows Explorer și pe bara de sarcini.

VEDEȚI...

► Pentru mai multe detalii privind înregistrarea pictogramelor aplicațiilor, vedeți pagina 546.

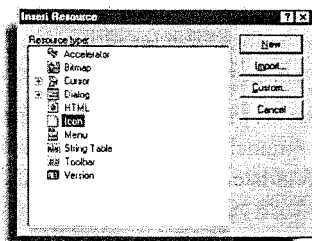
Inserarea unei noi resurse de tip pictogramă

Un proiect poate conține oricâte resurse de tip pictogramă. Acestea pot fi create și desenate de la zero sau importate dintr-un proiect existent sau din fișiere .ico. Pictogramele nou create sunt inițial transparente și sunt destinate implicit dispozitivelor VGA, având dimensiunea de 32x32. Pentru crearea unei noi pictograme urmați pașii de mai jos.

Crearea unei noi resurse de tip pictogramă

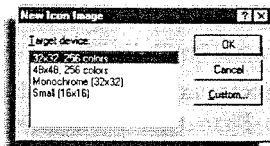
1. Apăsați Ctrl+R sau selectați **Resource** din meniul **Insert**. Va fi afișată caseta de dialog Insert Resource, înfățișată în figura 11.2.

FIGURA 11.2
Caseta de dialog Insert Resource.



2. Selectați **Icon** din lista **Resource Type** și apoi efectuați un clic pe **New**. Va fi creată o pictogramă goală, deschisă în editorul de resurse. Rețineți că puteți apăsa Ctrl+4 ca scurtătură pentru crearea unei noi pictograme.
3. Selectați dimensiunea imaginii. Pentru a selecta o dimensiune implicită, utilizați caseta combinată **Devices**.
4. Pentru a adăuga o nouă dimensiune de imagine, efectuați un clic pe butonul **New Device Image (Ins)** aflat în dreapta casetei combinate **Devices**; va fi afișată caseta de dialog New Icon Image, ilustrată în figura 11.3.

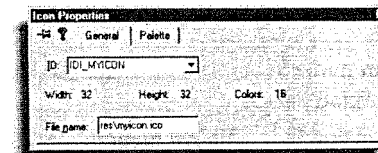
FIGURA 11.3
Caseta de dialog New Icon Image.



5. Selectați dimensiunea de imagine dorită din caseta cu listă **Target Device** sau efectuați un clic pe **Custom** pentru a specifica exact lățimea, înălțimea și numărul de culori ale imaginii.
6. Selectați **Properties** din meniul **View**. Este afișată caseta de dialog Icon Properties, înfățișată în figura 11.4. Această casetă de dialog cu proprietăți poate fi afișată și efectuând un dublu clic oriunde pe suprafața ferestrei editorului de imagine, mai puțin în interiorul imaginii.
7. Introduceți un nume în caseta combinată **ID**. Identificatorii asociați pictogramelor sunt de regulă prefixați cu **IDI_**.

8. Introduceți un nume de fișier în caseta de editare **File Name**. Rețineți că acest pas este opțional, deoarece Developer Studio generează un nume unic pentru fiecare fișier pictogramă, aceste fișiere fiind plasate în subdirectorul /res din directorul principal al proiectului.

FIGURA 11.4
Caseta de dialog Icon Properties.



Pentru a înlătura o pictogramă dintr-un proiect, selectați identificatorul acesteia în cadrul paginii ResourceView și apăsați tasta Delete. Aceasta nu va șterge fișierul .ico de pe disc, operație care trebuie efectuată manual. Pentru a elimina o pictogramă destinată unui anumit dispozitiv, selectați dimensiunea de imagine în caseta combinată **Devices**, după care selectați **Delete Device Image** din meniul **Image**.

Inserarea unei noi resurse de tip bitmap

Resursele de tip bitmap au numeroase întrebuințări. Pot fi utilizate pentru efecte vizuale, așa cum ați văzut deja bitmap-ul cu steagul în carouri afișat în ultimul pas din AppWizard. Este posibilă, de asemenea, afișarea unei imagini bitmap pe un buton de comandă, ca alternativă la o etichetă, lucru despre care vom discuta mai târziu în acest capitol.

Spre deosebire de dimensiunile relativ limitate ale pictogramelor, o imagine bitmap poate avea până la 2048x2048 de pixeli. Din nou spre deosebire de pictograme, bitmap-urile nu conțin atribute de culoare transparentă sau inversă. Un proiect poate conține oricâte resurse bitmap, acestea putând fi create de la zero sau importate dintr-un proiect existent sau dintr-un fișier (.bmp). Pentru a crea un nou bitmap, urmați pașii din secvența „Crearea unei noi resurse de tip bitmap”.

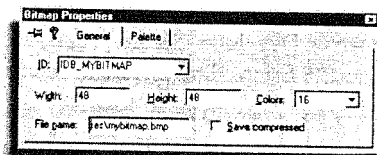
Pentru a înlătura un bitmap dintr-un proiect, selectați identificatorul bitmap-ului din pagina ResourceView și apăsați tasta Delete. Aceasta nu va duce și la ștergerea fișierului .bmp de pe disc, operație care trebuie efectuată manual.

Crearea unei noi resurse de tip bitmap

1. Apăsați Ctrl+R sau selectați **Resource** din meniul **Insert**. Va fi afișată caseta de dialog Insert Resource (vedeți figura 11.2).
2. Selectați **Bitmap** din lista **Resource Type** și apoi efectuați un clic pe **New**. Va fi creat un bitmap gol, deschis în editorul de resurse. Rețineți că puteți apăsa Ctrl+5 ca scurtătură pentru crearea unui nou bitmap.
3. Selectați **Properties** din meniul **View**; va fi afișată caseta de dialog Bitmap Properties, ilustrată în figura 11.5. Rețineți că această casetă de dialog cu proprietăți poate fi afișată și efectuând un dublu clic oriunde pe suprafața ferestrei editorului de imagine, mai puțin în interiorul imaginii.

FIGURA 11.5

Caseta de dialog Bitmap Properties.



1. Introduceți un nume în caseta combinată **ID**. Identificatorii asociați bitmap-urilor sunt de regulă prefixați cu **IDB_**.
2. Stabiliți dimensiunile imaginii și numărul de culori. Pentru detalii privind această operație, consultați secțiunea care urmează, „Modificarea dimensiunilor și numărului de culori ale unui bitmap”.
3. Introduceți un nume de fișier în caseta de editare **File Name**. Acest pas este opțional, deoarece Developer Studio generează un nume unic pentru fiecare fișier bitmap, aceste fișiere fiind plasate în subdirectorul /res.

Modificarea dimensiunilor și numărului de culori ale unui bitmap

O resursă nouă de tip bitmap are în mod implicit 16 culori și dimensiunea de 48x48 pixeli. Pentru a modifica dimensiunea sau numărul de culori, urmați secvența de pași „Modificarea proprietăților unui bitmap”. La reducerea dimensiunii sunt eliminați pixelii din marginile dreaptă și inferioară. Prin mărirea dimensiunii, la marginile dreaptă și inferioară sunt adăugați pixeli umpluți cu culoarea curentă de fundal.

Modificarea proprietăților unui bitmap

1. În cadrul paginii ResourceView, expandați catalogul **Bitmap** și efectuați un dublu clic pe identificatorul bitmap-ului pe care doriți să-l editați.
2. Selectați **Properties** din meniul **View**. Va fi afișată caseta de dialog Bitmap Properties. Rețineți că această casetă de dialog cu proprietăți poate fi afișată și efectuând un dublu clic oriunde pe suprafața ferestrei editorului de imagine, mai puțin în interiorul imaginii.
3. Efectuați un clic pe pagina **General**. Introduceți lățimea (**Width**) și înălțimea (**Height**) dorite. Dimensionarea unui bitmap se poate face și prin tragerea nodurilor de dimensionare afișate în editorul de imagine pe chenarul ce încadrează bitmap-ul.
4. Selectați numărul de culori dorit în caseta combinată **Colors**.
5. Pentru a personaliza paleta de culori, efectuați un clic pe pagina **Palette** și apoi efectuați un dublu clic pe culoarea pe care doriți să o modificați. Va fi afișată caseta de dialog Custom Color Selector, înfățișată în figura 11.6. Selectați culoarea dorită și apoi efectuați clic pe **OK**.

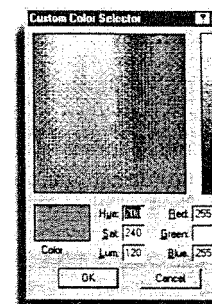
Importarea imaginilor

Dacă aveți talent, probabil că veți prefera să concepeți și să creați propriile imagini. Puteți, pe de altă parte, să importați resurse din fișiere și proiecte existente, sau chiar din fișiere executabile aflate în sistem. Pentru a importa o pictogramă sau un bitmap dintr-un fișier .ico

sau .bmp, urmați secvența de pași „Importarea unei resurse dintr-un fișier”. Asamblarea fișierului executabil al proiectului va implica și încorporarea resurselor. Prin urmare, resursele pot fi extrase și din fișierele executabile create de dumneavoastră sau de alții.

FIGURA 11.6

Caseta de dialog Custom Color Selector.



Importarea unei resurse dintr-un fișier

1. Selectați **Import** din meniul contextual al paginii ResourceView, sau selectați **Resource** din meniul **Insert** și apoi efectuați un clic pe butonul **Import** din caseta de dialog Insert Resource. Va fi afișată caseta de dialog Import Resource. Această casetă de dialog este o versiune puțin modificată a dialogului File Open.
2. Accesați directorul corespunzător și selectați fișierul(-ele) conținând resursa(-ele) pe care doriți să o (le) importați; apoi efectuați un clic pe butonul **Import**. Puteți utiliza caseta combinată **File of Type** pentru a determina afișarea exclusiv a unui anumit tip de fișiere resursă.
3. Dacă fișierul selectat are un tip recunoscut, resursa va fi adăugată la proiect. Va fi generat automat un identificator unic și în directorul proiectului va fi plasată o copie a fișierului importat.

Importarea resurselor din fișierele executabile

1. Selectați **Open** din meniul **File**. Va fi afișată caseta de dialog Open obișnuită.
2. Selectați **Resources** din caseta combinată **Open As**.
3. Accesați directorul corespunzător și selectați fișierul executabil (.exe, .dll sau .ocx) din care doriți să importați; apoi efectuați un clic pe butonul **Open**. Resursele încapsulate în fișierul executabil vor fi afișate în fereastra editorului (vedeți figura 11.7). Ca exemplu, selectați fișierul Cards.dll, plasat în mod normal în directorul C:\Windows\System. La deschiderea fișierelor din sistem este bine să validați opțiunea **Open as Read-Only**.
4. Pentru vizualizarea resurselor, deschideți caseta de dialog cu proprietăți și efectuați un clic pe pioneta aflată în colțul din stânga sus pentru a menține caseta deschisă. Cadrul **Preview** va înfățișa fiecare resursă selectată.
5. Pentru a insera o resursă din lista afișată, selectați-o cu mouse-ul. Țineți butonul mouse-ului apăsat împreună cu tasta **Ctrl** și deplasați cursorul în interiorul paginii

ResourceView. Eliberați butonul mouse-ului în momentul în care alături de cursorul său este afișat un semn plus. Resursa selectată va fi adăugată la proiect.

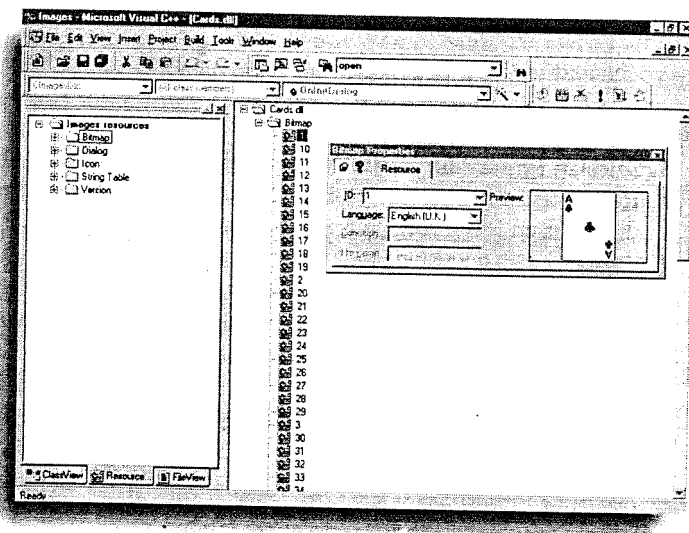
6. Din meniul **File**, selectați **Close**.

Utilizarea imaginilor în casetele de dialog

Există multe modalități de a afișa imagini în cadrul casetelor de dialog. Pictogramele și bitmap-urile sunt afișate cu ușurință într-o casetă de dialog, folosind un control imagine și stabilindu-i proprietățile pentru a indica resursa imagine dorită. Controalele imagine pot fi utilizate și pentru crearea unor dreptunghiuri colorate sau a unor cadre, cu scopul de a grupa elementele dintr-o casetă de dialog.

FIGURA 11.7

Importarea resurselor din fișierele executabile.



Capitolul de față se concentrează asupra utilizării resurselor imagine. Pentru afișarea altor imagini în zona client a unei casete de dialog trebuie să supradefiniți funcția `OnPaint`. Despre aceasta vom discuta în detaliu în capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”.

VEDEȚI...

➤ Pentru amănunte privind utilizarea funcției `OnPaint()`, vedeți pagina 330.

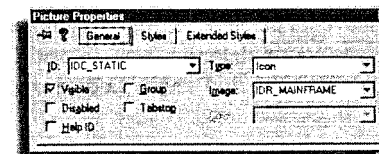
Stabilirea proprietăților unui control imagine

AppWizard adaugă automat la proiect resursa pictogramă `IDR_MAINFRAME`, care conține o reprezentare MFC tridimensională. Pentru a afișa această pictogramă, AppWizard plasează în caseta de dialog About un control imagine. Figura 11.8 înfățișează proprietățile acestui control. Casetă combinată **Type** specifică tipul imaginii și, dacă este selectată una din opțiunile **Icon** sau

Bitmap, identificatorul de resursă este specificat în caseta combinată **Image**. Tabelul 11.1 descrie tipurile de imagini disponibile pentru afișarea într-un control imagine.

FIGURA 11.8

Casetă de dialog Picture Properties.



TABELUL 11.1 Tipuri de imagini acceptate de controlul imagine

Tip	Descriere
Cadru	Afișează un cadru negru, alb, gri sau adâncit. Folosit ca efect vizual pentru gruparea elementelor.
Casetă	Afișează o casetă neagră, albă, gri sau tridimensională.
Pictogramă	Afișează o resursă imagine de tip pictogramă.
Bitmap	Afișează o resursă imagine de tip bitmap.
Metafișier extins	Afișează imaginea stocată într-un metafișier extins.

Ce este un metafișier extins?

Un metafișier stochează o imagine într-un format independent de dispozitiv. Pe lângă imaginea propriu-zisă, un metafișier conține date ce descriu imaginea din punct de vedere al dimensiunii, al obiectelor grafice folosite și al paletelor de culori. Formatul de metafișier extins este folosit exclusiv în aplicațiile Win32.

Încărcarea resurselor imagine la momentul execuției

Pentru a încărca o resursă pictogramă sau bitmap la momentul execuției, atașați o variabilă `CStatic` la controlul imagine. Pentru aceasta va trebui să schimbați identificatorul implicit `IDC_STATIC` al controlului. Tabelul 11.2 prezintă funcțiile membru ale clasei `CStatic` care se folosesc pentru a specifica imaginea afișată de un control. Pentru fiecare există un echivalent `Get` care întoarce imaginea afișată.

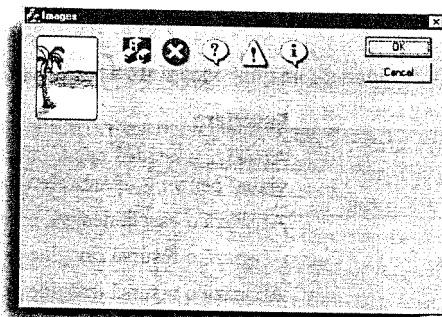
TABELUL 11.2 Funcțiile clasei `CStatic`

Funcție	Descriere
<code>SetIcon</code>	Stabilește pictograma care va fi afișată
<code>SetBitmap</code>	Stabilește bitmap-ul care va fi afișat
<code>SetCursor</code>	Stabilește cursorul care va fi afișat
<code>SetEnhMetaFile</code>	Stabilește metafișierul extins care va fi afișat

Înainte de a putea apela vreuna din funcțiile `set` prezentate în tabelul 11.2, imaginea resursă trebuie să fie în prealabil încărcată. Modul de încărcare a unei imagini depinde

de tipul acesteia. Figura 11.9 înfățișează exemplul Images, care încarcă la momentul execuției un bitmap și mai multe pictograme. Dacă doriți să încercați acest exemplu, apăsați la AppWizard pentru a crea o aplicație de tip dialog numită Images. Odată creat proiectul, urmați secvența de pași „Adăugarea unui control imagine într-o casetă de dialog”. După crearea controalelor imagine, folosiți ClassWizard pentru a atașa la fiecare câte o variabilă `CStatic`, urmând secvența de pași „Atașarea unei variabile la un control imagine”.

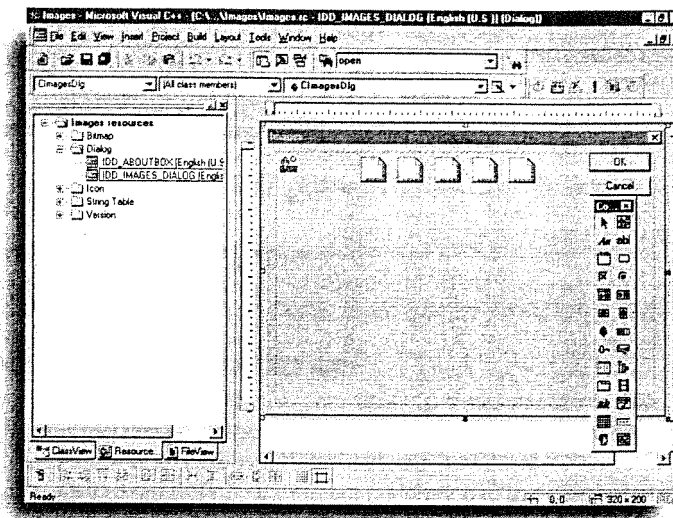
FIGURA 11.9
Exemplul Images.



Adăugarea unui control imagine într-o casetă de dialog

1. În cadrul paginii ResourceView, expandați catalogul **Dialog** și efectuați un dublu clic pe identificatorul casetei de dialog pe care doriți să o editați; pentru exemplul nostru, aceasta va fi `IDD_IMAGES_DIALOG`. Efectuați un clic pe eticheta **TODO: Place Dialog Box Controls Here** și apăsați tasta **Delete** pentru a o înlătura.
2. Selectați pictograma controlului imagine din bara cu instrumente Controls și plasați un astfel de control în colțul stânga sus al casetei de dialog.
3. Selectați **Properties** din meniul **View**. Va fi afișată caseta de dialog Picture Properties. Puteți menține această casetă de dialog deschisă efectuând un clic pe pioniza din colțul stânga sus.
4. Introduceți un nume în caseta combinată **ID**; pentru exemplul nostru, introduceți `IDC_B1`.
5. Selectați în caseta combinată **Type** tipul imaginii care va fi afișată de către control; pentru acest prim control selectați **Bitmap**.
6. Selectați din nou controlul imagine din bara cu instrumente Controls și adăugați încă cinci controale la dreapta primului, așa cum se vede în figura 11.10.
7. Selectați toate cele cinci controale adăugate, ținând apăsată tasta **Ctrl** și efectuând clic pe fiecare control. Odată selectate toate cinci, selectați opțiunea **Icon** în caseta combinată **Type**.
8. Selectați pe rând fiecare control imagine și specificați-i numele în caseta combinată **ID**; pentru exemplul nostru, introduceți `IDC_I1`, `IDC_I2` etc.

FIGURA 11.10
Aspectul casetei de dialog
Images.



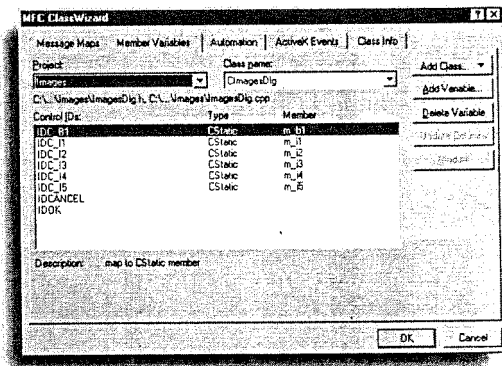
Atașarea unei variabile la un control imagine

1. Apăsați **Ctrl+W** pentru a apela ClassWizard sau selectați **ClassWizard** din meniul **View**.
2. Selectați pagina **Member Variables**.
3. Selectați `CImagesDlg` din caseta combinată **Class Name**.
4. Selectați `IDC_B1` în caseta cu listă **Control IDs**, după care efectuați un clic pe butonul **Add Variable**. Va fi afișată caseta de dialog Add Member Variable. Adăugarea unei variabile membru asociată cu un control se poate face și prin efectuarea unui dublu clic pe identificatorul controlului respectiv.
5. Introduceți numele variabilei în caseta **Member Variable Name**; în acest caz, introduceți `m_b1`.
6. Selectați **Control** în caseta combinată **Category**. Efectuați un clic pe **OK**.
7. Repetați pașii 4-6 pentru fiecare dintre controalele de la `IDC_I1` până la `IDC_I5`, atribuind fiecărei variabile un nume unic; specificați `m_i1`, `m_i2` și tot așa.
8. Caseta de dialog ClassWizard ar trebui să arate acum ca în figura 11.11.

Deoarece primul control urmează să afișeze un bitmap, va trebui să adăugați în proiect o resursă bitmap. Figura 11.9 înfățișează un bitmap importat din fișierul `cards.dll`. Pașii care trebuie urmați pentru a importa acest bitmap sunt descriși în secvența „Importarea resurselor din fișiere executabile”, prezentată în secțiunea „Importarea imaginilor”, mai devreme în acest capitol. Imaginea utilizată în exemplul nostru are numărul 64. Odată importat bitmap-ul, stabiliți-i identificatorul `IDB_BEACH`.

FIGURA 11.11

Caseta de dialog MFC ClassWizard.



Un bitmap poate fi încărcat prin intermediul unui obiect CBitmap, apelând funcția LoadBitmap a acestuia și transmițându-i identificatorul resursei. Pentru început, adăugați în clasa CImagesDlg o variabilă membru de tip CBitmap (fie m_bmp numele variabilei). Acum adăugați codul prezentat în listingul 11.1 în funcția OnInitialUpdate, după comentariile TODO.

LISTINGUL 11.1 LST12_1.CPP – Încărcarea resurselor imagine și configurarea controalelor imagine

```

1 // ** Incarcam resursa bitmap si
2 // ** atasam la controlul imagine bitmap-ul incarcat
3 VERIFY(m_bmp.LoadBitmap(IDB_BEACH)); — ❶
4 m_b1.SetBitmap(m_bmp); — ❷
5
6 CWinApp* pApp = AfxGetApp();
7 HICON hIcon;
8
9 // ** Incarcam pictograma aplicatiei si
10 // ** atasam la controlul imagine pictograma incarcata
11 hIcon = pApp->LoadIcon(IDR_MAINFRAME); — ❸
12 m_i1.SetIcon(hIcon);
13
14 // ** Incarcam cateva pictograme standard si
15 // ** le atasam la fiecare dintre controalele imagine
16 hIcon = pApp->LoadStandardIcon(IDI_HAND); — ❹
17 m_i2.SetIcon(hIcon);
18
19 hIcon = pApp->LoadStandardIcon(IDI_QUESTION);
20 m_i3.SetIcon(hIcon);
21
22 hIcon = pApp->LoadStandardIcon(IDI_EXCLAMATION);

```

(continuare)

LISTINGUL 11.1 Continuare

```

23 m_i4.SetIcon(hIcon);
24
25 hIcon = pApp->LoadStandardIcon(IDI_ASTERISK);
26 m_i5.SetIcon(hIcon);

```

- ❶ LoadBitmap() este un membru al clasei CBitmap.
- ❷ SetBitmap() atașează bitmap-ul la controlul imagine reprezentat de m_b1.
- ❸ Încarcă pictograma MFC a aplicației.
- ❹ LoadStandardIcon încarcă pictograme standard.

Apelul LoadBitmap din linia 3 încarcă resursa bitmap IDB_BEACH și o atașează la variabila m_bmp de tip CBitmap. Obiectul CBitmap este apoi transmis funcției CStatic::SetBitmap, în linia 4, indicând astfel controlului asociat cu m_b1 bitmap-ul care trebuie afișat.

Funcția CWinApp::LoadIcon apelată în linia 11 încarcă resursa pictogramă IDR_MAINFRAME și întoarce un HICON (indicator către un obiect grafic de tip pictogramă). Acest indicator este transmis apoi funcției CStatic::SetIcon, indicând controalelor respective ce pictograme să afișeze.

Prin intermediul funcției CWinApp::LoadStandardIcon sunt încărcate o serie de pictograme standard, așa cum se vede în liniile 16, 19, 22 și 25. Transmiteți identificatorul pictogramei dorite și funcția întoarce un HICON.

Acum asamblați și rulați aplicația. Caseta de dialog ar trebui să arate ca în figura 11.9.

Crearea butoanelor grafice

Deoarece imaginile sunt adesea mai ușor de perceput decât cuvintele, uneori este de preferat afișarea unei imagini pe suprafața unui buton, în locul unei etichete textuale. Tocmai în acest scop, biblioteca MFC pune la dispoziție clasa CBitmapButton. Se folosește editorul de imagini pentru a crea bitmap-urile, iar butoanele de comandă sunt plasate într-o machetă de dialog în maniera cunoscută. La crearea de resurse pentru butoanele grafice trebuie să țineți cont de două lucruri. În primul rând, specificați un șir de caractere ca identificator al bitmap-ului, și nu un număr care să fie folosit apoi cu #define. În al doilea rând, selectați proprietatea Owner Draw a controlului buton de comandă. Resursa bitmap va fi asociată cu butonul de comandă prin intermediul funcției CBitmapButton::AutoLoad.

Apelați la AppWizard pentru a crea o aplicație de tip dialog numită Smile. În cadrul ei veți crea propriile bitmap-uri care vor fi afișate pe un buton de comandă. Imaginea butonului se va schimba odată cu modificarea stării acestuia – de exemplu, la efectuarea unui clic.

Mai întâi înlăturați controlul etichetă TODO din macheta de dialog IDD_SMILE_DIALOG. Acum adăugați un buton de comandă în centrul casetei de dialog. Specificați IDC_SMILE

ca identificator și SMILE ca etichetă. Rețineți că dimensiunea butonului nu este importantă, deoarece acesta se va adapta la dimensiunile bitmap-ului.

Crearea de bitmap-uri pentru stările butonului

Un buton de comandă poate avea patru stări (ridicat, apăsător, dezactivat sau selectat). Starea butonului se modifică atunci când utilizatorul efectuează un clic sau apasă tasta Tab, dar se poate schimba și prin cod. Butoanele de comandă normale, care conțin etichete textuale, se ocupă automat de modificarea aspectului odată cu starea. De pildă, eticheta este afișată adâncită atunci când butonul este dezactivat.

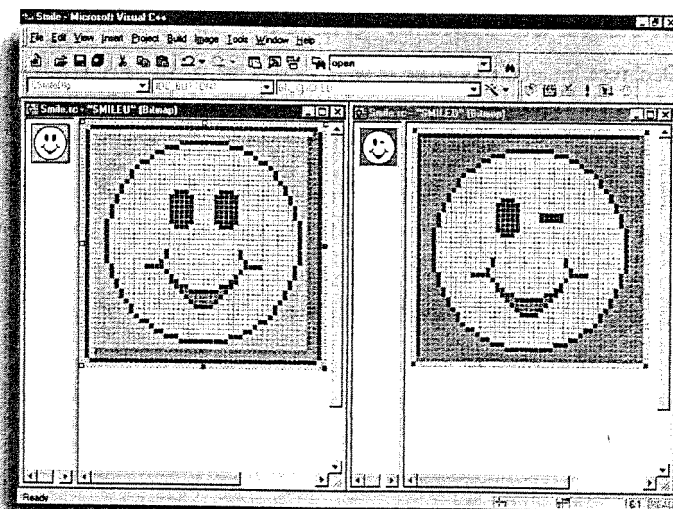
De afișarea butoanelor grafice este responsabil cel care le controlează; prin urmare, dumneavoastră sunteți responsabil de furnizarea unei imagini pentru fiecare stare. În cazul în care puneți la dispoziție o singură imagine, utilizatorul nu va vedea nici o reacție atunci când efectuează un clic pe buton. Trebuie, așadar, să furnizați imagini măcar pentru stările ridicat și apăsător. Identificatorul unui astfel de bitmap trebuie să fie un șir de caractere (inclusiv ghilimelele) bazat pe eticheta butonului. De exemplu, dacă un buton are eticheta „SMILE”, identificatorii celor patru imagini vor fi “SMILEU” (ridicat), “SMILED” (apăsător), “SMILEX” (dezactivat) și “SMILEF” (selectat). De fapt, acestea sunt și cele patru bitmap-uri necesare în exemplul nostru; pentru crearea lor urmați pașii de mai jos.

Crearea imaginilor pentru butoanele grafice

1. Apăsați Ctrl+5. Va fi creat un bitmap gol, afișat în editorul de resurse.
2. Selectați **Properties** din meniul **View**. Va fi afișată caseta de dialog Bitmap Properties.
3. Introduceți “SMILEU” (inclusiv ghilimelele) în caseta combinată **ID**.
4. Editați bitmap-ul pentru a-l face similar cu cel ilustrat în figura 11.12.

FIGURA 11.12

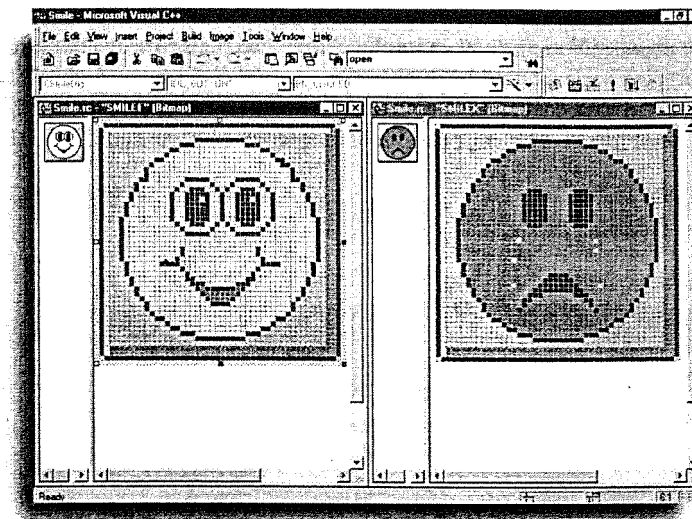
Bitmap-uri pentru stările ridicat și apăsător.



5. Repetați pașii 1-4 pentru a crea încă trei imagini bitmap.
6. Specificați numele de fișiere pentru aceste trei bitmap-uri, stabilindu-le identificatorii “SMILED”, “SMILEF” și “SMILEX”. Editați-le pe fiecare pentru a arăta similar cu imaginile prezentate în figurile 11.12 și 11.13.

FIGURA 11.13

Bitmap-uri pentru stările selectat și dezactivat.



Utilizarea clasei asociate butoanelor grafice

Așa cum am precizat mai devreme, biblioteca MFC pune la dispoziție o clasă CBitmapButton pentru asocierea de bitmap-uri cu butoanele de comandă. Această clasă este derivată din CButton și nu are decât câteva funcții membru. Este implementată funcția virtuală DrawItem, apelată pentru controalele desenate de către posesor. Singura funcție pe care o veți apela este AutoLoad.

Adăugați în clasa CSmileDlg o variabilă membru de tip CBitmapButton. Specificați m_bbSmile ca nume al variabilei. Acum adăugați în funcția OnInitUpdate, după comentariile // TODO:, următoarea linie de cod:

```
VERIFY(m_bbSmile.AutoLoad(IDC_SMILE, this);
```

Macrodefiniția VERIFY se folosește pentru a testa codul întors de funcția AutoLoad; dacă acesta este FALSE, în modul depanare va fi încălcată o aserțiune. Puteți observa că funcția AutoLoad primește identificatorul de resursă IDC_SMILE al butonului. Pe baza acestuia va determina eticheta butonului și apoi va încărca bitmap-urile ale căror nume se bazează pe respectiva etichetă.

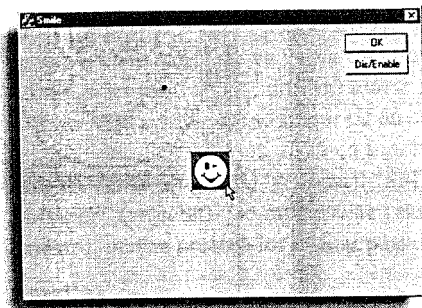
Înainte de a asambla programul mai trebuie făcut un singur lucru. Pentru a testa programul, trebuie să găsiți o cale de a trece butonul în starea dezactivat. Puteți face acest lucru modificând butonul **Cancel**. Mai întâi înlocuiți eticheta acestuia cu **Dis/Enable**, după care

schimbați identificatorul cu `ID_DISABLE`. Acum adăugați o funcție pentru tratarea mesajului `BN_CLICKED` provenit de la butonul `ID_DISABLE`. Ar trebui să fie creată o funcție membru numită `OnDisable`. Pentru ca această funcție să comute starea de activare, după comentariile // TODO: adăugați următoarea linie de cod:

```
m_bbSmile.EnableWindow( !m_bbSmile.IsWindowEnabled() );
```

Acum asamblați și rulați exemplul `Smile`. Încercați să selectați butonul grafic apăsând tasta `Tab`; veți vedea cum aspectul acestuia se modifică atunci când primește și pierde focusul. De asemenea, încercați să comutați starea de activare. Dacă efectuați un clic și țineți apăsat butonul mouse-ului, aspectul butonului ar trebui să fie cel înfățișat în figura 11.14.

FIGURA 11.14

Exemplul `Smile`.

VEDEȚI...

- Pentru mai multe informații despre macrodefiniția `VERIFY`, vedeți pagina 642.
- Pentru informații ajutătoare privind adăugarea funcțiilor de tratare, vedeți pagina 72.

Utilizarea imaginilor în cadrul controalelor

Capitolul de față a discutat deja despre utilizarea imaginilor bitmap împreună cu butoanele de comandă, dar există și alte câteva controale care permit folosirea directă a imaginilor: controlul listă, controlul arbore și noul control casetă combinată extinsă. Toate aceste controale întrețin o listă de elemente. Fiecare element are de regulă un atribut textual, dar poate fi asociat și cu două imagini, una pentru starea ne-selectat și cealaltă pentru starea selectat. Aceste imagini pot fi bitmap-uri sau pictograme.

Despre listele de imagini

Imaginile afișate de către o listă, un arbore sau o casetă combinată trebuie să se afle într-o listă de imagini. O listă de imagini este încapsulată într-un obiect `CImageList`. Se creează un astfel de obiect și apoi se adaugă în cadrul său bitmap-uri sau pictograme. Toate imaginile trebuie să aibă aceleași dimensiuni. După aceea se apelează funcția membru `SetImageList` a controlului în cauză, transmitând un pointer la obiectul `CImageList`. La inserarea unui element în control, acesta este asociat cu o imagine din lista de imagini prin specificarea indicelui acesteia, pornind de la zero. Pentru că lista de imagini se află într-un obiect separat de control, este posibil ca mai multe controale să partajeze o aceeași listă de imagini.

Modificarea listelor de imagini

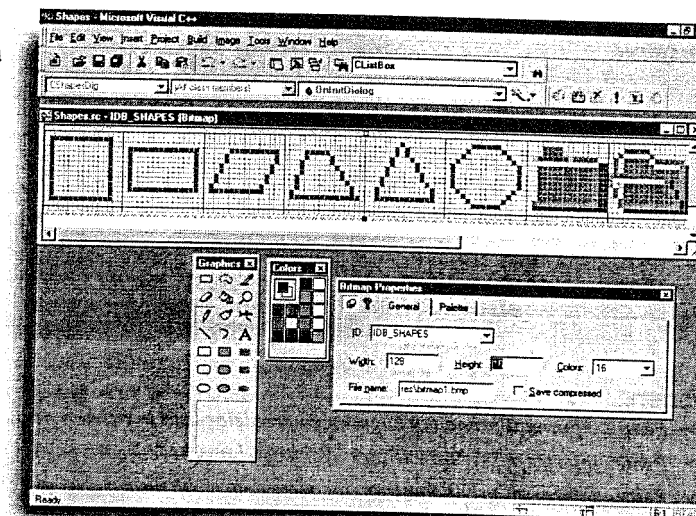
Aveți posibilitatea de a adăuga, înlocui și înlocui imagini dintr-o listă de imagini și de a activa desenarea transparentă prin specificarea unei măști.

Pentru a vedea exact cum se construiește o listă de imagini și cum se utilizează ea într-un control, apelați la `AppWizard` și creați un proiect de tip dialog numit `Shapes`. După crearea proiectului, următorul lucru de care aveți nevoie sunt câteva imagini. Atunci când doriți să adăugați mai multe imagini într-o listă, soluția cea mai simplă este să creați un bitmap care să conțină toate imaginile. Exemplul nostru va folosi opt imagini; ele vor fi stocate într-un singur bitmap, ilustrat în figura 11.15.

Creați o nouă resursă de tip bitmap, stabilind identificatorul (**ID**) `IDB_SHAPES`, o lățime (**Width**) de 118 pixeli și o înălțime (**Height**) de 16 pixeli. Apoi editați bitmap-ul pentru a-i da aspectul din figura 11.15. Dacă nu sunteți sigur în legătură cu pașii care trebuie urmați, consultați secțiunile „Inserarea unei noi resurse de tip bitmap” și „Utilizarea editorului de imagini”, aflate mai devreme în acest capitol.

FIGURA 11.15

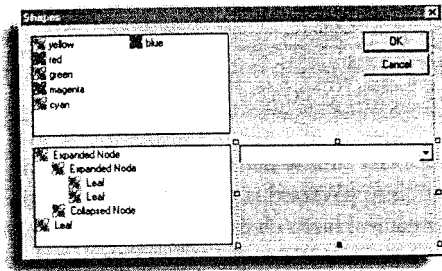
Bitmap-ul pregătit pentru lista de imagini.



Acum adăugați în dialogul `IDD_SHAPES_DIALOG` o listă, un arbore și o casetă combinată, ca în figura 11.16. Între proprietățile controlului listă, introduceți `IDC_LIST1` în câmpul **ID** și selectați **List** din caseta combinată **View** aflată în pagina **Styles**. Între proprietățile controlului arbore, introduceți `IDC_TREE1` în câmpul **ID** și validați opțiunile **Has Buttons**, **Has Lines** și **Lines at Root** din pagina **Styles**. În ceea ce privește proprietățile casetei combinate, introduceți `IDC_COMBOBOXEX1` în câmpul **ID** și selectați **Drop List** din caseta combinată **Type** aflată în pagina **Styles**.

FIGURA 11.16

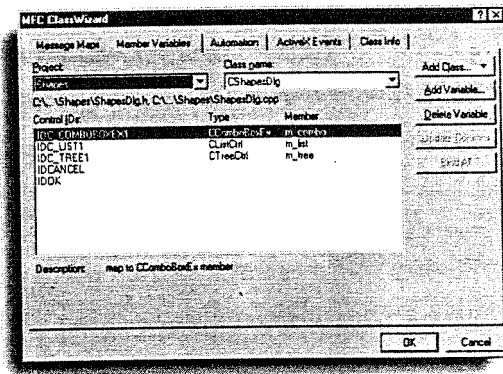
Aspectul casetei de dialog Shapes.



Acum folosiți ClassWizard pentru a adăuga în clasa CShapeDlg câte o variabilă membru pentru fiecare control. Specificați `m_list`, `m_tree` și `m_combo` ca nume ale variabilelor și selectați de fiecare dată opțiunea **Control** din caseta combinată **Category** a casetei de dialog Add Member Variable. Odată adăugate variabilele, pagina **Member Variables** din ClassWizard ar trebui să arate ca în figura 11.17.

FIGURA 11.17

Caseta de dialog MFC ClassWizard.



VEDEȚI...

- Pentru informații privind adăugarea controalelor listă în casetele de dialog, vedeți pagina 114.
- Pentru informații privind utilizarea controalelor arbore, vedeți pagina 441.
- Pentru informații privind utilizarea controalelor listă, vedeți pagina 427.

Crearea și utilizarea unei liste de imagini

O listă de imagini conține o mulțime de imagini bitmap folosite în cadrul unor controale. Lista de imagini este încapsulată într-un obiect `CImageList`. De exemplu, adăugați în clasa `CShapeDlg` o variabilă de tip `CImageList` numită `m_imageList`. Întreg codul necesar pentru crearea listei de imagini și pentru inserarea elementelor în cele trei controale poate fi plasat în interiorul funcției `OnInitDialog`. Adăugați după comentariile `// TODO:` din `OnInitDialog` codul prezentat în listingul 11.2.

LISTINGUL 11.2 LST12_2.CPP – Crearea și utilizarea unei liste de imagini

```

1 // ** Cream etichete textuale pentru imaginile cu figuri
2 static char* shape[] = {
3     "Square", "Rectangle", "Parallelogram",
4     "Trapezoid", "Triangle", "Octagon"};
5 int nShapes = 6;
6
7 // ** Incarcam resursa bitmap
8 CBitmap bitmap;
9 VERIFY(bitmap.LoadBitmap(IDB_SHAPES)); — 1
10
11 // ** Cream lista de imagini si adaugam bitmap-ul
12 m_imageList.Create(IDB_SHAPES, 16, 1, 0); — 2
13 m_imageList.Add(&bitmap, (COLORREF)0xFFFFFFFF);
14
15 // ** Configuram controlul lista pentru a utiliza lista de imagini
16 m_list.SetImageList(&m_imageList, LVSIL_SMALL); — 3
17
18 // ** Inseram 6 elemente in controlul lista,
19 // ** specificand un indice de imagine
20 for(int i = 0; i < nShapes; i++)
21     m_list.InsertItem(i, shape[i], i);
22
23 // ** Configuram controlul arbore pentru a utiliza lista de imagini
24 m_tree.SetImageList(&m_imageList, TVSIL_NORMAL); — 4
25
26 // ** Inseram doua noduri radacina in arbore si alegem ca imagini
27 // ** de nod normal/selectat un dosar inchis/deschis
28 HTREEITEM hTetrasons, hOther;
29 hTetrasons = m_tree.InsertItem("Tetrasons", 6, 7);
30 hOther = m_tree.InsertItem("Other", 6, 7);
31
32 // ** Inseram primele 3 figuri ca subordonate nodului 'Tetrasons'
33 // ** si ultimele 3 ca subordonate nodului 'Other'
34 for(i = 0; i < nShapes; i++)
35     m_tree.InsertItem(shape[i], i, i,
36         i < 3 ? hTetrasons : hOther);
37
38 // ** Configuram caseta combinata pentru a utiliza lista de imagini
39 m_combo.SetImageList(&m_imageList);
40
41
42 // ** Alocam o structura de tip element si stabilim masca
43 COMBOBOXEXITEM cbItem;

```

```

43 cbItem.mask = CBEIF_TEXT|CBEIF_IMAGE|CBEIF_SELECTEDIMAGE; — 5
44
45 // ** Inseram 6 elemente in caseta combinata
46 // ** stabilim imaginile de nod normal/selectat la aceeasi imagine
47 for(i = 0; i < nShapes; i++)
48 {
49     cbItem.pszText = shape[i];
50     cbItem.iItem = i;
51     cbItem.iImage = cbItem.iSelectedImage = i; — 6
52     m_combo.InsertItem(&cbItem);
53 }

```

- 1 Încărcăm resursa bitmap IDB_SHAPES.
- 2 Creăm o listă de imagini, specificând o lățime de 16 pixeli pentru fiecare imagine.
- 3 Configurăm controlul listă pentru a folosi lista de imagini creată.
- 4 Lista de imagini este partajată de controlul arbore.
- 5 Precizăm care dintre membrii structurii COMBOBOXITEM sunt semnificativi la efectuarea apelului InsertItem().
- 6 Inițializăm structura COMBOBOXITEM și apoi inserăm elementul.

La prima vedere, listingul 11.2 ar putea părea puțin cam complex, dar de fapt nu sunt decât 28 de linii de cod – restul sunt comentarii.

În linia 2 se creează un vector de șiruri de caractere, numit `shape`. Aceste șiruri corespund primelor șase imagini din bitmap-ul `IDB_SHAPES`. În liniile 8 și 9 se folosește un obiect `CBitmap` local pentru încărcarea resursei `IDB_SHAPES`; observați că funcția `VERIFY` duce la încălcarea unei aserțiuni în cazul în care încărcarea resursei eșuează.

Obiectul `m_imagelist` de tip `CImageList` este inițializat în linia 12 printr-un apel `Create`. Primul parametru reprezintă identificatorul resursei, iar cel de-al doilea este lățimea în pixeli a fiecărei imagini. Al treilea parametru specifică o valoare de creștere a listei, utilizată pentru optimizarea performanțelor. Prin ultimul parametru se poate transmite o mască de culoare care să permită transparență la afișarea imaginilor. Odată inițializată lista de imagini, imaginile propriu-zise sunt adăugate prin apelul `Add` (linia 13). Aici este transmisă adresa bitmap-ului, fiind astfel adăugate toate cele opt imagini.

Crearea listei de imagini

Sunt disponibile mai multe versiuni supradefinite ale funcției `CImageList::Create`. Acestea vă pot permite să transmiteți un șir conținând identificatori de resurse sau să concatenați două liste de imagini existente pentru a crea una nouă.

După inițializarea listei de imagini, controalele pot fi îndrumate să o folosească. Operația este similară pentru toate controalele, fiind efectuată prin apelurile `SetImageList` din liniile 16, 24 și 39. Pe lângă adresa listei de imagini, controalele listă și arbore primesc un

parametru suplimentar care precizează tipul listei de imagini. Tabelele 11.3 și 11.4 prezintă tipurile posibile.

Inițializarea listelor de imagini

Pentru adăugarea unei pictograme într-o listă de imagini există o versiune supradefinită a funcției `Add` care primește un parametru de tip `HICON`.

Pentru a adăuga imagini direct dintr-un fișier de pe disc, apelați funcția `Read`.

TABELUL 11.3 Tipurile de liste de imagini acceptate de un control listă

Tipul listei de imagini	Descrierea tipului
<code>LVSIL_NORMAL</code>	Listă de imagini conținând pictograme mari
<code>LVSIL_SMALL</code>	Listă de imagini conținând pictograme mici
<code>LVSIL_STATE</code>	Listă de imagini conținând imagini de stare

TABELUL 11.4 Tipurile de liste de imagini acceptate de un control arbore

Tipul listei de imagini	Descrierea tipului
<code>TVSIL_NORMAL</code>	Listă de imagini conținând imagini pentru noduri selectate și ne-selectate
<code>LVSIL_STATE</code>	Listă de imagini conținând imagini de stare definite de utilizator

Inserarea elementelor în controlul listă se realizează în bucla `for` din linia 20. Funcția `InsertItem` primește trei parametri. Primul dintre aceștia reprezintă poziția elementului în cadrul controlului, al doilea este conținutul textual al elementului, iar cel de-al treilea specifică poziția unei imagini în cadrul listei de imagini. În cazul de față, valoarea `i` transmisă este incrementată astfel încât fiecărui element îi este asociată următoarea imagine din listă.

Controlul arbore conține două noduri rădăcină care au fiecare câte trei descendenți. Cele două elemente rădăcină sunt inserate în liniile 29 și 30. În apelurile `InsertItem` este transmis conținutul textual al elementului ca prim parametru. Al doilea parametru precizează poziția imaginii folosite atunci când elementul nu este selectat (în acest caz, 6 corespunde unui dosar închis). Al treilea parametru specifică poziția imaginii folosite atunci când elementul este selectat (în acest caz, 7 corespunde unui dosar deschis).

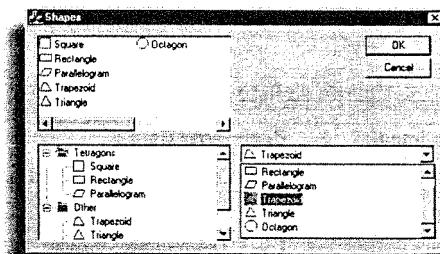
Elementele subordonate sunt inserate în cadrul controlului arbore prin intermediul buclei `for` din linia 34. Funcția supradefinită `InsertItem` primește aici patru parametri. Primul reprezintă conținutul textual al elementului. Al doilea și al treilea sunt pozițiile imaginilor folosite atunci când elementul nu este selectat, respectiv atunci când este selectat (în cazul de față este folosită o aceeași imagine pentru ambele stări). Ca al patrulea parametru se transmite `hTetragons` sau `hOther`, specificând astfel părintele noului element.

Elementele din caseta combinată sunt inserate într-o manieră diferită. Aici este necesară completarea unei structurii `COMBOBOXITEM` (`cbItem`) cu date despre element, transmițând

apoi adresa acesteia în apelul funcției `InsertItem`. Prin fixarea indicatorului `mask` din linia 43 se precizează care dintre elementele structurii sunt valide sau, altfel spus, ce variabile din structură sunt completate. Așa cum vedeți în linia 51, variabilele `iImage` și `iSelectedImage` sunt ambele fixate la aceeași poziție de imagine.

Asamblați și rulați proiectul. Figura 11.18 înfățișează programul Shapes.

FIGURA 11.18
Programul Shapes.



CAPITOLUL

12

Utilizarea documentelor, a reprezentărilor și a cadrelor

Crearea unei aplicații cu interfață de tip document singular

**Dezvoltarea unei aplicații pe baza arhitecturii MFC
Document/Reprezentare**

Despre clase document, reprezentare, cadru și machetă de document

Adăugarea de noi elemente în resursa meniu standard

Miliarde de dolari cheltuite la dezvoltarea automobilelor a oferit acestora o interfață cu utilizatorul remarcabil de universală. Datorită acestui lucru puteți trece dintr-un tip de mașină într-un altul, nefiind necesare mai mult de câteva minute pentru a vă familiariza. Odată ce ați învățat elementele de bază ale conducerii auto în fața unui volan, puteți să aplicați lesne cunoștințele dobândite asupra oricărui automobil, dispozitivele de control fiind practic identice, lăsând la o parte diversele cosmetizări.

Popularitatea aplicațiilor Windows provine din conceptul de interfață universală. O aplicație Windows tipică dispune de o bară de titlu în partea superioară, urmată de un meniu derulant și de unul sau mai multe rânduri de bare cu instrumente ale căror pictograme au rol de scurtături. Sub barele cu instrumente se află zona principală de afișare, iar la baza ferestrei se găsește o bară de stare. Chiar și între mai multe tipuri diferite de aplicații, aceleași opțiuni de meniu și pictograme de pe bara cu instrumente sunt identice și își păstrează semnificația. În acest fel este posibil să treceți de la programele de prelucrare a textelor la aplicații de poștă electronică, operând voios asupra datelor în doar câteva minute.

Pentru programatorul în Visual C++, conformarea la aspectul și atmosfera Windows se face relativ fără efort. Biblioteca MFC oferă un sprijin substanțial la generarea unei noi aplicații prin intermediul AppWizard, care poate să creeze automat un meniu, o bară cu instrumente, o bară de stare și alte componente, permițând apoi personalizarea ușoară a fiecărui element. Clasele create cu AppWizard lucrează împreună pentru a forma o structură omogenă cunoscută sub numele de *arhitectură Document/Reprezentare* (Document/View).

Conceptul fundamental al arhitecturii Document/Reprezentare este separarea datelor propriu-zise de prezentarea a ceea ce datele semnifică pentru utilizator. Aceasta se realizează stocând datele într-o clasă (clasa document) și informațiile privind prezentarea într-o altă clasă (clasa reprezentare), de unde și denumirea de Document/Reprezentare.

Adevărata semnificație a termenului „document”

Prima aplicație care a utilizat tehnica de programare Document/Reprezentare a fost MS Word, rolul său principal fiind crearea de documente cu conținut textual și grafic. Dar termenul *document* nu este limitat la această semnificație convențională. Clasa Document poate conține orice tip de date, ceea ce face ca arhitectura Document/Reprezentare să fie adaptabilă la diverse aplicații.

Există două categorii de aplicații Document/Reprezentare: SDI (Single Document Interface – interfață de tip document singular) și MDI (Multiple Document Interface – interfață de tip document multiplu). Capitolul de față se concentrează asupra aplicațiilor SDI.

VEDEȚI...

➤ Pentru mai multe informații despre aplicațiile MDI, vedeți pagina 490.

Crearea unei aplicații SDI

Crearea unei aplicații care să conțină toate elementele vizuale de interfață familiare din Windows, cum ar fi o bară cu instrumente sau o bară de stare, ar fi destul de dificilă dacă AppWizard nu s-ar afla la dispoziția dumneavoastră. De fapt, AppWizard face mai mult

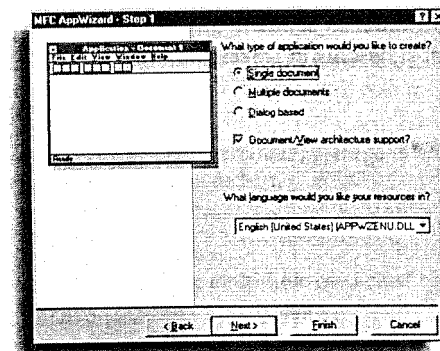
decât să ofere elementele de bază; există și o serie de opțiuni suplimentare. Spre exemplu, capacitatea de a conecta aplicația la o bază de date este o astfel de opțiune; o alta privește implementarea unei funcționalități elementare de poștă electronică. În secțiunea de față veți crea o aplicație SDI care va fi utilizată pentru a inspecta clasele generate și pentru a vedea cum operează arhitectura Document/Reprezentare. Întâi de toate, creați un nou proiect numit SDICoin.

Crearea unei aplicații SDI cu AppWizard

1. Selectați **New** din meniul **File**. În cadrul paginii **Projects** a casetei de dialog **New** selectați **MFC AppWizard (exe)**.
2. Specificați un nume pentru proiect (**SDICoin**, în cazul de față) și apoi efectuați un clic pe **OK**.
3. În caseta de dialog **MFC AppWizard Step 1**, selectați butonul de opțiune **Single Document** și asigurați-vă că opțiunea **Document/View Architecture Support?** este validată (vedeți figura 12.1). Efectuați un clic pe **Next**.

FIGURA 12.1

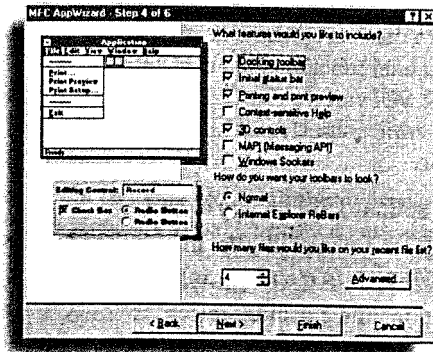
Pasul 1 din AppWizard.



4. În caseta de dialog a pasului 2, selectați butonul de opțiune **None**. Opțiunile prezente aici se referă la diferite tipuri de suport pentru bazele de date, nefiind necesare pentru exemplul nostru. Efectuați un clic pe **Next**.
5. În caseta de dialog a pasului 3, selectați butonul de opțiune **None** și validați opțiunea **ActiveX Controls**. Celelalte opțiuni din fereastră privesc adăugarea unor facilități de automatizare OLE care nu sunt necesare pentru exemplul nostru. Efectuați un clic pe **Next**.
6. În caseta de dialog a pasului 4, validați următoarele opțiuni: **Docking Toolbar**, **Initial Status Bar**, **Printing and Print Preview** și **3D Controls**. Selectați butonul de opțiune **Normal** pentru stilul de bară cu instrumente. Consultați figura 12.2 pentru a confirma opțiunile selectate. Efectuați un clic pe **Next**.
7. În caseta de dialog a pasului 5, selectați **MFC Standard** ca stil al proiectului, selectați **Yes** pentru a genera comentarii în fișierele sursă și selectați **As a Shared DLL** pentru a indica modul de utilizare a bibliotecii MFC. Efectuați un clic pe **Next**.

FIGURA 12.2

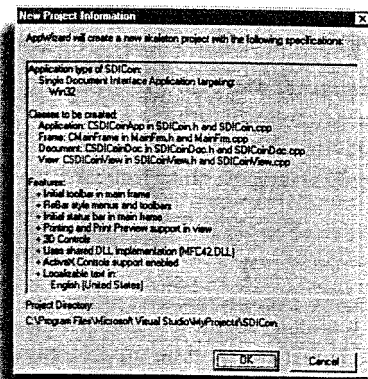
Pasul 4 din AppWizard.



8. În caseta de dialog a pasului 6 efectuați un clic pe **Finish**. Va fi afișată caseta de dialog New Project Information, ca în figura 12.3. Această casetă de dialog conține câteva informații utile privind clasele generate și numele fișierelor sursă. Tot aici puteți confirma lista de facilități. Efectuați un clic pe **OK**. Noul proiect ar trebui să fie acum creat și deschis în Developer Studio.

FIGURA 12.3

Caseta de dialog New Project Information.



Aplicația pe care o veți dezvolta va afișa un teanc de monede. Opțiunile de meniu vor permite adăugarea sau înlăturarea unei monede din teanc. Datele, care sunt de fapt numărul de monede, sunt reținute în clasa document, fiind accesate de clasa reprezentare în scopul afișării teancului. Deși exemplul este unul relativ simplu, ar trebui să vă ofere o imagine asupra scopului fundamental al arhitecturii Document/Reprezentare, și anume *încapsularea* datelor. Prin încapsulare, datele sunt stocate exclusiv în clasa document, fiind oferite funcții de acces care să permită clasei reprezentare să prezinte informațiile către utilizator în orice formă dorită. De exemplu, în loc să deseneze un teanc de monede, clasa reprezentare ar putea fi determinată să afișeze pur și simplu numărul de monede fără ca clasa document să sufere vreo modificare. Dar înainte de a începe dezvoltarea proiectului sunt necesare câteva explicații privind clasele generate de AppWizard.

O aplicație cu interfață de tip document singular conține o clasă document derivată din clasa MFC CDocument și o clasă reprezentare care poate fi derivată din oricare dintre clasele reprezentare MFC. Diferitele clase reprezentare reflectă tipurile de aplicații care pot fi dezvoltate în jurul arhitecturii Document/Reprezentare. Clasa reprezentare de bază este selectată în pasul 6 din AppWizard. Tabelul 12.1 înfățișează clasele disponibile; exemplul SDICoin folosește clasa CView implicită.

Documentele încapsulează datele aplicației

Clasa document este derivată din CDocument și variabilele sale membru conțin datele aplicației. Funcțiile membru ale clasei pe care o derivați din CDocument vor oferi modalități de manipulare a datelor. De pildă, funcția Serialize() este folosită pentru încărcarea și stocarea datelor pe disc.

TABELUL 12.1 Clasele reprezentare de bază oferite de AppWizard

Numele clasei	Descriere
CView	Clasa implicită care implementează funcționalitatea esențială a unei reprezentări, inclusiv interacțiunea cu clasa document.
CScrollView	Derivată din CView și dezvoltată pentru a permite derularea în cazurile în care suprafața logică a reprezentării este mai mare decât suprafața afișabilă efectiv.
CListView	Folosește un control listă ca reprezentare, permițând aplicației să afișeze pictograme sau coloane de text.
CTreeView	Folosește un control arbore ca reprezentare, permițând aplicației să afișeze informații ierarhice.
CEditView	Folosește un control de editare multi-linie ca și clasă reprezentare, adăugând facilități de derulare și de căutare.
CRichEditView	Folosește un control de editare formatată ca reprezentare, oferind facilități de editare mai puternice decât clasa CEditView.
CFormView	Folosește o machetă de dialog ca reprezentare, permițând aplicației să se prezinte ca un formular de bază de date.
CRecordView	Derivată din CFormView și dezvoltată pentru a face legătura între reprezentare și o mulțime de înregistrări dintr-o bază de date. Această clasă este disponibilă numai dacă a fost selectat suportul pentru baze de date.
CWebView	Folosește reprezentarea Internet Explorer pentru a permite crearea unei aplicații browser de Web.

VEDEȚI...

- Pentru informații privind clasa CListView, vedeți pagina 427.
- Pentru informații privind clasa CTreeView, vedeți pagina 441.
- Pentru informații privind clasa CRichEditView, vedeți pagina 449.
- Pentru informații privind clasa CWebView, vedeți pagina 454.
- Pentru informații privind clasa CScrollView, vedeți pagina 415.
- Pentru informații privind clasele CFormView și CRecordView, vedeți pagina 576.

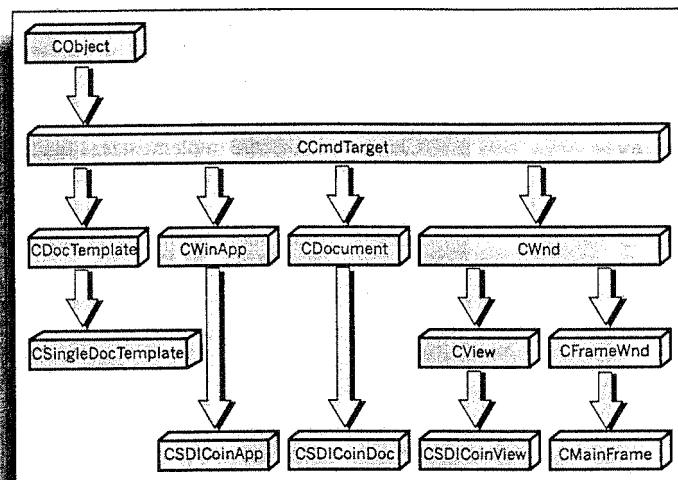
- Pentru informații privind manipularea fișierelor în cadrul arhitecturii Document/Reprezentare, vedeți pagina 532.
- Pentru informații privind adăugarea suportului pentru baze de date, vedeți pagina 568.
- Pentru informații privind automatizarea OLE, vedeți pagina 589.

Despre clasele unei aplicații SDI

Structura de clase a unei aplicații cu interfață de tip document singular diferă de cea a unei aplicații de tip dialog. Dacă priviți pagina ClassView a proiectului SDICoin, puteți observa că AppWizard a creat patru clase care gestionează structura aplicației. Lanțul de moștenire pentru aceste clase este ilustrat în figura 12.4.

FIGURA 12.4

Ierarhia de clase pentru o aplicație SDI.



Clasa CSDICoinApp, derivată din CWinApp, joacă mai multe roluri și diferă în implementare de obiectul aplicație dintr-un proiect de tip dialog. Această clasă se ocupă de inițializarea aplicației. Este responsabilă pentru întreținerea relației dintre clasele document, reprezentare și cadru. De asemenea, primește mesaje Windows și le expediază către fereastra țintă corespunzătoare.

CObject este stră-străbunic

Clasa CObject este clasa de bază a aproape tuturor claselor din biblioteca MFC. Toate clasele derivate din CObject moștenesc capacitatea de a verifica la momentul execuției tipul de clasă al unui obiect, de a furniza informații de diagnosticare și de a suporta serializare.

Așa cum am precizat mai devreme, într-o aplicație cu interfață de tip document singular, la un moment dat va exista o singură instanță a clasei CSDICoinDoc derivată din CDocument. Rolul clasei CSDICoinDoc este acela de container pentru datele aplicației. Aceste date pot avea orice formă; în exemplul Coin este vorba de un simplu contor, dar de obicei datele

necesare pentru a satisface cerințele unei aplicații sunt mult mai complexe. În cazul unei aplicații de prelucrare a textului, datele ar include conținutul textual al documentului, dar și informațiile de formatare corespunzătoare, în timp ce datele dintr-o aplicație de proiectare grafică ar trebui să conțină coordonate și alte detalii ale fiecărei componente ale unei imagini.

Modul în care se implementează stocarea datelor în cadrul documentului este responsabilitatea programatorului. În mod ideal, trebuie aleasă o abordare orientată pe obiect care să permită păstrarea datelor complet separate de interfața cu utilizatorul. Respectarea acestei tehnici aduce codului atât flexibilitate, cât și portabilitate. În exemplul Coin există o singură reprezentare, dar ați putea avea nevoie de mai multe, toate folosind aceleași date din document. Reprezentările ar putea avea tipuri diferite; în caz că da, trebuie implementat cu multă grijă mecanismul folosit de clasa document pentru a oferi date fiecărei reprezentări.

Clasa CSDICoinView este responsabilă de a furniza către utilizator o reprezentare a datelor documentului și de a permite acestuia să interacționeze cu respectivele date. O reprezentare nu poate fi interfață decât pentru o singură instanță a unui document, în timp ce un document poate avea ca interfețe mai multe obiecte reprezentare. Clasa reprezentare accesează datele necesare prin intermediul funcțiilor membru oferite de către document.

Rolul clasei CMainFrame este acela de a furniza aplicației o fereastră. Clasa din care se derivează este CFrameWnd, aceasta fiind încapsularea unei ferestre simple. În mod implicit, printre indicatorii de stil ai ferestrei se numără WS_OVERLAPPEDWINDOW, care dotează fereastra cadru cu o bară de titlu conținând butoane de minimizare/maximizare și închidere, un meniu sistem și capacitatea de deplasare și dimensionare. Tot printre indicatorii de stil se află și FWS_ADDTOTITLE, care face ca numele documentului încărcat să fie afișat automat pe bara de titlu. Clasa CMainFrame se ocupă, de asemenea, de crearea, inițializarea și distrugerea barelor cu instrumente și a barei de stare.

Modificarea stilului unei ferestre cadru

Puteți să modificați opțiunile de stil ale unei ferestre cadru înainte de afișarea acesteia supradefinind funcția PreCreateWindow(). Această funcție primește o referință la o structură CREATESTRUCT care conține atributul style, pe care îl puteți modifica înainte de a fi transmis implementării din clasa de bază a funcției.

VEDEȚI...

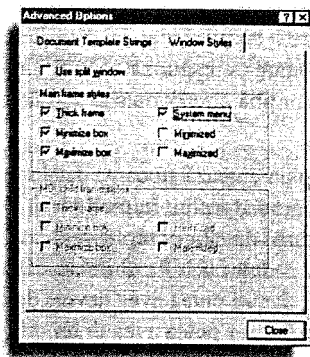
- Pentru a afla mai multe despre diferența dintre clasele SDI și MDI, vedeți pagina 490.

Elementele vizuale ale unei aplicații SDI

Acceptarea opțiunilor implicite din pasul 4 al procesului de creare a unei aplicații SDI cu AppWizard va duce la o aplicație având o bară de titlu cu butoane de minimizare/maximizare și închidere, un meniu derulant ce include opțiuni pentru tipărire și previzualizare, o bară cu instrumente și o bară de stare. Unele dintre aceste elemente sunt, însă, opționale. Priviți figura 12.2 pentru a vedea opțiunile disponibile în pasul 4. Prin efectuarea unui clic pe butonul **Advanced** din caseta de dialog a pasului 4 sunt oferite și alte opțiuni, ilustrate în figura 12.5. Acestea permit personalizarea stilului ferestrei cadru.

FIGURA 12.5

Opțiuni suplimentare oferite de AppWizard în pasul 4.



Zona client a ferestrei cadru nu este folosită pentru a implementa reprezentarea. Aceasta beneficiază de o fereastră proprie, care este o fereastră copil a ferestrei cadru, lipsită de margine și de meniuri. Această fereastră a reprezentării acoperă întreaga zonă client a ferestrei cadru (vedeți figura 12.6). Reprezentarea este astfel implementată pentru ca același model să poată fi utilizat atât în aplicațiile cu document singular, cât și în cele cu document multiplu, iar o aplicație MDI conține clase diferite pentru ferestrele cadru și copil. Fereastră cadru este cea care primește mesajele legate de meniuri și de fereastră în sine.

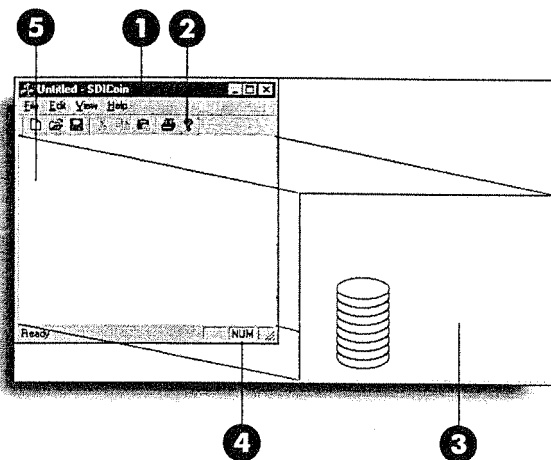
Mesaje pentru fereastră cadru

Anumite mesaje sunt trimise numai ferestrelor cadru. Printre acestea se află deplasarea, dimensionarea, minimizarea și maximizarea, și mesajele ce privesc afișarea zonei non-client a ferestrei aplicației.

FIGURA 12.6

Elementele vizuale ale unei aplicații SDI.

- 1 CMainFrame
- 2 CToolBar
- 3 CSDICoinView
- 4 CStatusBar
- 5 Fereastră reprezentării



În cele ce urmează sunt prezentate diferitele componente ale ferestrei:

- **CMainFrame**. Derivată din `CFrameWnd`; furnizează fereastra cadru a aplicației.
- **CToolbar**. Implementează barele cu instrumente ancorate de fereastra cadru.
- **CSDICoinView**. Implementează reprezentarea ca și copil al ferestrei cadru.
- **CStatusBar**. Implementează bara de stare atașată la fereastra cadru.
- **Fereastră reprezentării**. Se suprapune peste zona client a ferestrei cadru.

Poate că vă întrebați de unde vin facilități vizuale precum meniul și bara cu instrumente. Acestea sunt de fapt resurse pe care AppWizard le-a adăugat la proiect odată cu crearea sa. Dacă selectați pagina `ResourceView`, veți vedea că există patru resurse (o tabelă de acceleratori, o pictogramă, un meniu și o bară cu instrumente) care au același identificator, `IDR_MAINFRAME`. Fiecare dintre resurse poate fi editată efectuând un dublu clic pe identificatorul resursei. Există, în plus, o intrare specială în tabela de șiruri care are același identificator `IDR_MAINFRAME`. Aceasta reprezintă un șir de caractere cu un format special; șirul este împărțit în șapte șiruri distincte, delimitate de `\n` (linie nouă). Fiecare subșir descrie un anumit atribut al documentului aplicației. De exemplu, primul subșir reprezintă titlul afișat în fereastra aplicației.

Despre machetele de document SDI

Până acum, capitolul de față s-a concentrat asupra claselor și facilităților vizuale individuale dintr-o aplicație Document/Reprezentare. Toate aceste elemente sunt îmbinate și gestionate prin intermediul unei clase machetă de document. Dacă priviți figura 12.4, care înfățișează ierarhia claselor dintr-o aplicație SDI, veți putea remarca clasa `CSingleDocTemplate`, care este clasa machetă de document utilizată de aplicațiile cu interfață de tip document singular.

O instanță a clasei `CSingleDocTemplate` este creată și folosită în funcția `CSDICoinApp::InitInstance` (vedeți listingul 12.1). Funcția `InitInstance` este apelată de infrastructura MFC chiar la începutul execuției, cu scopul de a efectua inițializarea aplicației.

LISTINGUL 12.1 LST13_3.CPP – Funcția *InitInstance*

```
1 BOOL CSDICoinApp::InitInstance() — 1
2 {
3     // ** Observatie: unele linii au fost inlaturate pentru a reduce
4     // lungimea
5     // Register the application's document templates.
6     // Document templates
7     // serve as the connection between documents,
8     // frame windows and views.
9
10    CSingleDocTemplate* pDocTemplate;
```

(continuare)

LISTINGUL 12.1 Continuare

```

9   pDocTemplate = new CSingleDocTemplate(
10       IDR_MAINFRAME, — ❷
11       RUNTIME_CLASS(CSDICoinDoc),
12       RUNTIME_CLASS(CMainFrame),
13       // main SDI frame window
14       RUNTIME_CLASS(CSDICoinView));
15   AddDocTemplate(pDocTemplate);
16   // Parse command line for standard shell commands,
17   // DDE, file open
18   CCommandLineInfo cmdInfo;
19   ParseCommandLine(cmdInfo); — ❸
20 // Dispatch commands specified on the command line
21 if (!ProcessShellCommand(cmdInfo)) — ❹
22     return FALSE;
23
24 // The one and only window has been initialized,
25 // so show and update it.
26 m_pMainWnd->ShowWindow(SW_SHOW); — ❺
27 m_pMainWnd->UpdateWindow();
28 return TRUE;
29 }

```

- ❶ Apelată de MFC la începutul execuției programului.
- ❷ Macheta de document este inițializată pe baza identificatorului de resursă și a claselor document, cadru și reprezentare.
- ❸ Sunt analizați parametrii din linia de comandă.
- ❹ Prelucrăm parametrii din linia de comandă și creăm un nou document sau deschidem un fișier.
- ❺ Afișăm și actualizăm fereastra principală a aplicației.

Puteți vedea cum în linia 9 este creat un obiect `CSingleDocTemplate`, fiind transmiși patru parametri. Primul parametru este identificatorul de resursă `IDR_MAINFRAME`. În exemplul `SDICoin`, `IDR_MAINFRAME` identifică patru resurse distincte: pictograma aplicației, un meniu, o bară cu instrumente și o tabelă de acceleratori. Următorii trei parametri sunt toți pointeri la informații de clasă dinamice pentru clasele document, cadru și, respectiv, reprezentare. Pointerii sunt generați cu ajutorul macrodefiniției `RUNTIME_CLASS`. Acest lucru este posibil din cauză că `AppWizard` a prevăzut suport pentru crearea dinamică a acestor clase, incluzând macrodefinițiile `DECLARE_DYNCREATE` și `IMPLEMENT_DYNCREATE`.

Obiectele document, reprezentare și cadru propriu-zise nu sunt create în acest moment. Codul inițializează obiectul `CSingleDocTemplate` cu informații necesare acestuia pentru a putea să încarce resursele și să aloce la cerere documente, reprezentări și cadre. Clasa machetă de document este numită și generator de clase.

Metoda generatorului de clase

Clasa machetă de document este un exemplu de generator de clasă, adică de clasă care definește modul în care se instanțiază alte clase. Prin urmare, un generator de clase știe să genereze clase specifice aplicației.

Obiectul machetă de document propriu-zis este reținut în clasa aplicației. Apelul funcției `AddDocTemplate` (în linia 14) înregistrează obiectul machetă de document nou creat în cadrul clasei `CWinApp`.

Clasa `CWinApp` păstrează obiectul `CSingleDocTemplate` până când este distrusă, ceea ce se întâmplă doar la închiderea aplicației. În cadrul acestui proces de distrugere, clasa `CWinApp` eliberează orice memorie alocată pentru macheta de document, așa că nu trebuie să vă preocupați de acest aspect.

VEDEȚI...

➤ Pentru detalii privind informațiile de clasă dinamice, vedeți pagina 532.

Utilizarea funcțiilor oferite de infrastructura Document/Reprezentare

Poate că cel mai dificil aspect legat de înțelegerea – și implicit utilizarea – arhitecturii Document/Reprezentare este urmărirea secvenței de creare a obiectelor implicate și a apelurilor de funcții virtuale care au loc. Această dificultate nu se datorează complexității, ci faptului că o bună parte din implementare are loc în culise. Este un lucru bun, pentru că astfel infrastructura face un efort substanțial în locul dumneavoastră. Dar pentru a înțelege exact cum funcționează modelul Document/Reprezentare este necesară o privire în spatele scenei.

După crearea și inițializarea obiectului `CSingleDocTemplate`, între liniile 8 și 14 ale listingului 12.1, are loc prelucrarea parametrilor din linia de comandă și apoi este afișată fereastra aplicației.

Clasa `CCommandLineInfo` (linia 17) ajută la tratarea parametrilor transmiși aplicației prin intermediul liniei de comandă. Funcția `CWinApp::ParseCommandLine` (linia 18) primește ca parametru o referință la obiectul `CCommandLineInfo`. Pentru fiecare parametru din linia de comandă este apelată funcția `ParseParam`, completându-se variabilele membru ale obiectului `CCommandLineInfo`. Tabelul 12.2 prezintă opțiunile standard din linia de comandă și scopul acestora.

Analiza propriilor parametri din linia de comandă

Cea mai simplă cale de a analiza parametrii proprii din linia de comandă este să derivați o clasă din `CCommandLineInfo` și să supradefiniți funcția `ParseParam`.

TABELUL 12.2 Parametrii din linia de comandă

Parametru	Scop
nimic	Se creează un nou document.
fișier	Se deschide fișierul specificat.
/p fișier	Se tipărește fișierul specificat la imprimanta implicită.
/pt fișier imprimantă port	Se tipărește fișierul specificat la imprimanta precizată.
/dde	Începe o sesiune DDE.
/automation	Lansează aplicația ca server de automatizare OLE.
/embedding	Lansează aplicația pentru a edita un obiect OLE încapsulat într-o altă aplicație.

Puteți vedea din tabelul 12.2 că dacă în linia de comandă nu este transmis nici un parametru, la lansarea aplicației este creat un nou document. Evident, aceasta se întâmplă în mod implicit. Pe lângă document, sunt create fereastra cadru și reprezentarea. Totul se întâmplă în funcția `CWinApp::ProcessShellCommand`, apelată în linia 21. Listingul 12.2 oferă o imagine de ansamblu asupra secvenței de evenimente implicate. Vă atrag atenția că întreg codul din listingul 12.2 este parafrizat și a fost condensat pentru a scoate în evidență procesul de creare.

LISTINGUL 12.2 LST13_4.CPP – Secvența de creare a documentului, reprezentării și cadrului

```

1  BOOL CWinApp::ProcessShellCommand( — ❶
    ↳ CCommandLineInfo& rCmdInfo)
2  {
3      if(rCmdInfo.NewFile)
4      {
5          OnFileNew();
6      }
7      if(rCmdInfo.OpenFile)
8      {
9          OpenDocumentFile(rCmdInfo.strFileName);
10     }
11     ...
12 }

13 void CWinApp::OnFileNew()
14 {
15     m_pDocManager->OnFileNew();
16 }

17 void CDocManager::OnFileNew()
18 {
19     pTemplate->OpenDocumentFile(NULL); — ❷

```

```

20 }

21 CSingleDocTemplate::OpenDocumentFile
    ↳ (LPCTSTR lpszPathName)
22 {
23     if(m_pOnlyDoc != NULL)
24         pDocument->SaveModified(); — ❸
25     else
26     {
27         pDocument = CreateNewDocument();
28         pFrame = CreateNewFrame(pDocument); — ❹
29     }
30     ...
31     pDocument->OnNewDocument(); — ❺
32     InitialUpdateFrame(pFrame, pDocument); — ❻
33 }

```

- ❶ Apelată de funcția `InitInstance`.
- ❷ `pTemplate` este un pointer la obiectul `CSingleDocTemplate` deja înregistrat.
- ❸ Dacă documentul există, este reinițializat.
- ❹ Obiectele document și cadru sunt construite folosind informațiile de clasă dinamice din machetă. `CreateNewFrame` construiește și reprezentarea.
- ❺ Funcție virtuală care este supradefinită pentru a se efectua inițializarea documentului.
- ❻ Efectuează inițializarea ferestrei cadru prin apelul `LoadFrame`. Activează reprezentarea și trimite mesajul `WM_INITIALUPDATE`, care duce la apelarea funcției `CView::OnInitialUpdate`.

Funcția `CWinApp::ProcessShellCommand`, ilustrată între liniile 1 și 12, conține o serie de apeluri ale unor funcții de infrastructură, depinzând de rezultatul analizei eventualelor parametri din linia de comandă. Dacă nu a fost specificat nici un parametru, funcția va apela `CWinApp::OnFileNew` (linia 13). Aceeași funcție este apelată și ca urmare a selectării de către utilizator a opțiunii **New** din meniul **File**, legătura fiind stabilită automat de `AppWizard`.

Listingul 12.2 ilustrează logica aflată la baza creării unui nou document la lansare, dar diferențele față de deschiderea unui fișier nu sunt mari. Deosebirea se înregistrează la început, acolo unde `CWinApp::ProcessShellCommand` fie găsește fișierul transmis, fie solicită utilizatorului să selecteze un fișier în cadrul unei casete de dialog tipice. Odată localizat fișierul, calea și numele acestuia sunt transmise aceleiași funcții `CSingleDocTemplate::OpenDocumentFile` (linia 21), în loc de `NULL`, care ar duce la crearea unui nou document.

`CWinApp::OnFileNew` delegă imediat răspunderea către `CDocManager::OnFileNew` (linia 15), iar aceasta determină un pointer către obiectul `CSingleDocTemplate` creat anterior. Pe baza acestui pointer este apelată funcția `OpenDocumentFile`, transmițându-se `NULL` ca parametru. Aceasta din urmă este funcția în care se desfășoară operațiile efective.

CSingleDocTemplate::OpenDocumentFile

Obiectul document dintr-o aplicație SDI este creat o singură dată (la primul apel `OpenDocumentFile`). Apelurile următoare ale acestei funcții, ca replică la selectarea opțiunilor **New** sau **Open** din meniul **File**, reinițializează documentul deschis pentru ca acesta să poată fi folosit din nou. Procesul de reinițializare are loc în funcția `SaveModified`. Funcția verifică mai întâi dacă documentul curent a fost modificat. Clasa `CDocument`, care este baza tuturor claselor document, include o variabilă membru de tip `BOOL` care arată dacă documentul a fost modificat. Acest membru este activat prin intermediul funcției `SetModifiedFlag`.

În cazul în care documentul curent a fost modificat, utilizatorului i se propune salvarea documentului înainte de a merge mai departe. Caseta de mesaj permite utilizatorului să continue cu sau fără salvare sau să anuleze execuția funcției `OpenDocumentFile`. Documentul este salvat printr-un apel `OnSaveDocument` și este reinițializat prin apelul `DeleteContents`, ambele fiind descrise mai târziu în această secțiune.

Documente OLE

Dacă specificați în AppWizard că aplicația dumneavoastră este un server sau un container OLE, clasa document va fi derivată din una dintre aceste trei subclase ale `CDocument`: `COleDocument`, `COleLinkingDoc` sau `COleServerDoc`. Fiecare dintre aceste trei clase dezvoltă funcționalitatea celei precedente. În esență, ele aduc posibilitatea de a trata documentul ca o listă de obiecte OLE.

Dacă obiectul document trebuie creat, este apelată funcția `CreateNewDocument` (linia 27) a machetei de document. Aceasta folosește informațiile de clasă dinamice pentru a construi obiectul document specific aplicației. În plus, funcția adaugă noul document la lista documentelor curente. La crearea documentului se creează, de asemenea, cadrul și reprezentarea, aceasta prin intermediul unei alte funcții a machetei, `CreateNewFrame`.

Funcția `CreateNewFrame` (linia 28) folosește informațiile de clasă dinamice ale machetei pentru a construi ambele obiecte fereastră cadru și reprezentare. Odată creat cadrul, este apelată funcția `CFrameWnd::LoadFrame`, fiindu-i transmis identificatorul de resursă specificat în constructorul machetei. Dacă vă amintiți, acesta era `IDR_MAINFRAME`. `LoadFrame` încarcă fiecare resursă (meniu, bară cu instrumente, pictogramă, tabelă de acceleratori și șir de caractere) și le atașează la fereastra cadru. Rețineți că dacă încărcarea vreuneia dintre resurse eșuează, funcția `LoadFrame` va eșua și va produce un mesaj de avertizare: `Warning: CDocTemplate couldn't create a frame. Acum sunt construite toate obiectele, așa că documentele vor fi inițializate prin intermediul funcției OnNewDocument (linia 31), descrisă ulterior în această secțiune.`

În fine, în secvența de creare este apelată funcția `InitialUpdateFrame` a machetei (linia 32). Aceasta stabilește reprezentarea nou creată ca activă, după care trimite mesajul `WM_INITIALUPDATE` către copiii ferestrei cadru, printre care se află și fereastra reprezentării. La primirea acestui mesaj este apelată funcția virtuală `CView::OnInitialUpdate`. Supradefinirea acestei funcții se obișnuiește în scopul

efectuării unor inițializări specifice ale reprezentării. Rețineți că de regulă este necesar să asigurați și apelul versiunii din clasa de bază a funcției.

CDocument::OnNewDocument

Această funcție este apelată în timpul procesului de creare a obiectului document. Aceasta se poate întâmpla la lansarea aplicației sau ca răspuns la selectarea de către utilizator a opțiunii **New** din meniul **File**. Funcția apelează `DeleteContents` pentru a șterge orice date existente în document și fixează la `FALSE` indicatorul de modificare (arătând că documentul nu este modificat). În plus, este resetat șirul de caractere care conține numele fișierului document.

CDocument::OnOpenDocument

Această funcție este apelată în procesul de deschidere a unui fișier existent. Aceasta se poate întâmpla la lansarea aplicației, în cazul în care s-a precizat un fișier ca parametru în linia de comandă, sau ca răspuns la selectarea de către utilizator a opțiunii **Open** din meniul **File**. Funcția apelează `DeleteContents` pentru a șterge orice date existente în document. Implementarea implicită deschide fișierul și apoi apelează funcția `Serialize` pentru încărcarea datelor documentului. Odată încărcate datele, fișierul este închis și indicatorul de modificare este fixat la `FALSE` (arătând că documentul nu este modificat).

CDocument::OnSaveDocument

Această funcție este apelată în procesul de salvare a unui fișier. Apelul se efectuează ca răspuns la selectarea de către utilizator a uneia dintre opțiunile **Save** sau **Save As** din meniul **File**. Funcția este apelată, de asemenea, dacă utilizatorul decide salvarea documentului atunci când i se propune acest lucru la încercarea de închidere a unui document modificat. Există un singur parametru, un pointer la un șir de caractere constant care conține numele fișierului. Implementarea implicită deschide fișierul și apoi apelează funcția `Serialize` pentru salvarea datelor documentului. Odată salvate datele, fișierul este închis și indicatorul de modificare este fixat la `FALSE` (arătând că documentul nu este modificat).

CDocument::DeleteContents

Această funcție este apelată de `OnNewDocument`, `OnOpenDocument` și `OnCloseDocument`. Rolul său constă în ștergerea conținutului documentului curent. Implementarea implicită nu face nimic, pentru că datele din document sunt specifice aplicației. Trebuie să furnizați o supradefiniție a acestei funcții care să elibereze orice memorie alocată în cadrul documentului și să reinițializeze variabilele.

CDocument::OnCloseDocument

Această funcție este apelată la închiderea documentului curent. În cazul unei aplicații SDI, aceasta se întâmplă când utilizatorul optează pentru deschiderea unui document sau pentru crearea unui nou și la închiderea aplicației. Funcția închide mai întâi toate reprezentările atașate la document. Apelează apoi `DeleteContents`, după care distruge obiectul document.

VEDEȚI...

- Pentru amănunte privind automatizarea OLE, vedeți pagina 589.
- Pentru informații despre manipularea fișierelor și serializare, vedeți pagina 532.

Utilizarea împreună a documentelor și a reprezentărilor

Implementarea arhitecturii Document/Reprezentare oferă baza pentru o aplicație funcțională și scalabilă. Separarea codului care manipulează datele de cel care se ocupă de interfața cu utilizatorul are ca rezultat o aplicație modulară care este mai ușor de întreținut și îmbunătățit. Spre exemplu, prezentarea unor date existente sub o nouă formă ține doar de dezvoltarea unei noi clase reprezentare care va folosi interfața existentă a documentului pentru a accesa informațiile necesare.

Proiectarea interacțiunii dintre perechea de clase Document/Reprezentare are o importanță fundamentală. O problemă obișnuită, de care se lovesc chiar și programatorii experimentați, este de a decide precis care informații țin de document și care țin de reprezentare. Nu există reguli stabilite și totul depinde în bună măsură de tipul aplicației. În mod ideal, reprezentarea nu ar trebui să conțină date efective, ci ar trebui să le obțină de la document atunci când este necesar. Astfel, modificările datelor se efectuează toate în același loc, fiind anunțate apoi, în scopul actualizării, toate reprezentările atașate la document. Dar o astfel de abordare nu este întotdeauna convenabilă sau eficientă. De pildă, aplicațiile de editare a textului rețin de multe ori în clasa reprezentare o copie integrală sau parțială a datelor din cauză că documentul este accesat în mod frecvent (poate la fiecare apăsare de tastă), ceea ce influențează performanțele.

Vor apărea multe situații în care veți găsi potrivită o dublare a unor date în cadrul reprezentării. Nu este nici o problemă – sunteți liber să scrieți codul așa cum doriți, dar dublări inutile ale datelor vă vor complica sarcina de programator, așa că este bine să le evitați pe cât posibil.

Inițializarea datelor documentului

Clasa `CDocument` este o clasă abstractă, ceea ce înseamnă că trebuie să derivați pe baza ei o clasă document proprie. Membrii acestei clase vor stoca datele aplicației. Lăsând la o parte aplicațiile foarte simple, documentele necesită multe variabile membru, adesea sub forma unor vectori alocați sau a unor liste înlănțuite.

Clasele MFC de tip colecție

Biblioteca MFC pune la dispoziție mai multe clase ajutătoare de tip colecție în scopul implementării unor structuri de date complexe, printre acestea fiind `CArray` și `CObList`. Apelați, acolo unde este posibil, la această funcționalitate existentă.

Vă amintiți că într-o aplicație SDI obiectul document este construit o singură dată și apoi refolosit pentru alte documente. Prin urmare, constructorul documentului nu este întotdeauna locul potrivit pentru inițializarea variabilelor membru, deoarece acesta va fi apelat o singură dată și nu pentru fiecare nou document.

În secțiunea „Utilizarea funcțiilor oferite de infrastructura Document/Reprezentare”, ați văzut că funcția virtuală `DeleteContents` a documentului este apelată automat în `OnNewDocument`, `OnOpenDocument` și `OnCloseDocument`. Toate acestea sunt situații în care documentul curent se schimbă. `DeleteContents` este responsabilă pentru ștergerea conținutului documentului, așa că după efectuarea acestei operații se pot inițializa foarte bine membrii documentului.

Adăugarea în document a variabilelor membru

Clasa document este asemeni oricărei alte clase din proiect; în consecință, variabilele membru pot fi adăugate prin intermediul opțiunii `Add Member Variable` a meniului contextual din pagina `ClassView`. În cazul în care adăugați mai multe variabile, ați putea găsi mai convenabilă editarea directă a codului. Pentru a deschide fișierul antet al clasei document trebuie să efectuați un dublu clic pe numele clasei în cadrul paginii `ClassView`.

Pentru a menține integritatea datelor, variabilele documentului ar trebui să fie *variabile membru protejate*. O variabilă protejată nu poate fi modificată decât de funcțiile membru ale clasei din care face parte sau ale unei clase derivate din aceasta. Prevenind alterarea datelor documentului de către orice altă clasă, rămâne un singur punct de modificare a unei variabile; aceasta înlesnește informarea în consecință a reprezentărilor atașate la document. Pentru a permite reprezentărilor să vadă datele protejate ale documentului, clasa document trebuie să ofere funcții de acces care să întoarcă referințe la variabilele membru.

Protejarea variabilelor documentului

Este recomandat ca o clasă document să conțină *variabile membru protejate*, care să fie accesate prin intermediul unor funcții membru publice.

Acum aveți informațiile necesare privind arhitectura Document/Reprezentare, așa că este momentul să începem dezvoltarea exemplului `SDICoin`. În secțiunea de față vă veți ocupa de clasa document, `CSDICoinDoc`. Informația necesară în această clasă este numărul curent de monede, care poate fi stocat într-o singură variabilă membru. De fiecare dată când clasa reprezentare trebuie să deseneze teancul de monede, va avea nevoie să acceseze acest contor. Soluția cea mai simplă în acest sens ar fi o variabilă membru publică în cadrul documentului care să rețină contorul, permițând accesul direct al clasei reprezentare. Dezavantajul acestei abordări este faptul că devine posibilă alterarea valorii contorului de către codul clasei reprezentare, poate chiar accidental. Soluția preferată este folosirea unei variabile membru protejate care să rețină contorul, împreună cu oferirea de către clasa document a unei funcții de acces prin care clasa de reprezentare să poată obține valoarea contorului. Pentru adăugarea codului necesar urmați pașii de mai jos.

Implementarea stocării datelor documentului și a metodelor de acces

1. Selectați clasa `CSDICoinDoc` din cadrul paginii `ClassView`; apoi selectați `Add Member Variable` din meniul contextual. Va fi afișată caseta de dialog `Add Member Variable`.

2. Introduceți `int` în caseta de editare **Variable Type**. Introduceți `m_nCoins` în caseta de editare **Variable Name** și selectați butonul de opțiune **Protected Access**. Efectuați un clic pe **OK**.
3. Având clasa `CSDICoinDoc` încă selectată, alegeți **Add Member Function** din meniul său contextual.
4. Introduceți `int` în caseta de editare **Function Type**. Introduceți `GetCoinCount` în caseta de editare **Function Declaration** și selectați butonul de opțiune **Public Access**.
5. Introduceți în corpul funcției `GetCoinCount` următoarea linie de cod:

```
return m_nCoins;
```
6. Apăsați **Ctrl+W** pentru a apela **Class Wizard** sau selectați **Class Wizard** din meniul **View**.
7. Selectați pagina **Message Maps**.
8. Selectați `CSDICoinDoc` din caseta combinată **Class Name**.
9. Selectați `CSDICoinDoc` în caseta cu listă **Object IDs**.
10. Selectați **DeleteContents** din caseta cu listă **Messages** și apoi efectuați un clic pe butonul **Add Function**.
11. Efectuați un clic pe **OK** pentru a închide caseta de dialog **Class Wizard**.
12. Introduceți următoarea linie de cod după comentariile `TODO` din funcția `DeleteContents`:

```
m_nCoins = 1;
```

`CSDICoinDoc` conține acum o variabilă membru protejată, `m_nCoins`, care este inițializată în funcția `DeleteContents` supradefinită. Ați adăugat, de asemenea, o funcție membru de acces, `GetCoinCount`, care va fi utilizată de către reprezentare pentru a determina valoarea `m_nCoins`.

Accesarea datelor documentului din cadrul reprezentării

Pentru ca reprezentarea să poată extrage date ale documentului, ea trebuie să fie mai întâi capabilă să acceseze obiectul document. Infrastructura MFC se ocupă automat de acest aspect, adăugând în clasa reprezentare a aplicației o funcție `GetDocument`. Această funcție întoarce un pointer către obiectul document la care reprezentarea a fost atașată prin intermediul machetei de document. De fapt, infrastructura pune la dispoziție două funcții `GetDocument`. Veți vedea în exemplul nostru că clasa `SDICoinView` conține o implementare `GetDocument` în `SDICoinView.cpp` și o alta în `SDICoinView.h`, așa cum se ilustrează mai jos. Cele două funcții fac exact același lucru; singura diferență este că prima se folosește în versiunea pentru depanare a proiectului, în timp ce a doua este utilizată în versiunea finală. Motivul pentru această abordare este faptul că funcțiile inline sunt mai eficiente, iar această funcție este probabil să fie apelată frecvent în timpul rulării.

Utilizarea funcțiilor inline poate îmbunătăți performanțele

Cuvântul cheie `inline` determină compilatorul să însereze codul funcției acolo unde aceasta este apelată, în loc să efectueze apelul de funcție. Viteza de execuție crește deoarece sunt eliminate apelurile de funcții. Rețineți că nu este posibilă execuția pas cu pas a unei funcții inline la momentul depanării.

SDICoinView.cpp

```
#ifdef _DEBUG
CSDICoinDoc* CSDICoinView::GetDocument()
{
    ASSERT(m_pDocument->IsKindOf(
        RUNTIME_CLASS(CSDICoinDoc)));
    return (CSDICoinDoc*)m_pDocument;
}
#endif // _DEBUG
```

SDICoinView.h

```
#ifndef _DEBUG // debug version in SDICoinView.cpp
inline CSDICoinDoc* CSDICoinView::GetDocument()
{ return (CSDICoinDoc*)m_pDocument; }
#endif
```

Clasa `CSDICoinView` este responsabilă pentru afișarea teancului de monede. Codul care se ocupă efectiv de desenarea monedelor este relativ simplu și se află în funcția `CSDICoinView::OnDraw`. **App Wizard** a creat deja un schelet al acestei funcții pe care puteți să-l folosiți. Infrastructura MFC apelează funcția `OnDraw` de fiecare dată când reprezentarea trebuie generată pe ecran – de exemplu, la dimensionarea ferestrei cadru. Funcția `OnDraw` este apelată, de asemenea, pentru a gestiona operațiile necesare pentru tipărire și previzualizare. Contextul dispozitiv țintă transmis funcției diferă după destinația reprezentării. Editați funcția `CSDICoinView::OnDraw` ca în listingul 12.3.

LISTINGUL 12.3 LST13_1.CPP – Accesarea datelor documentului pentru generarea reprezentării

```
1 void CSDICoinView::OnDraw(CDC* pDC)
2 {
3     // ** Obținem un pointer la document
4     CSDICoinDoc* pDoc = GetDocument(); — ❶
5     ASSERT_VALID(pDoc);
6
7     // TODO: add draw code for native data here
8     // ** Salvăm pensula curentă
9     CBrush* pOldBrush = pDC->GetCurrentBrush(); — ❷
10 }
```

(continuare)

LISTINGUL 12.3 Continuare

```

11 // ** Cream o pensula plina de culoare galbena
12 CBrush br;
13 br.CreateSolidBrush( RGB(255,255,0) ); — ③
14
15 // Selectam pensula galbena in contextul dispozitiv
16 pDC->SelectObject(&br); — ④
17
18 // Obtinem de la document numarul de monede si
19 // desenam cate doua elipse care sa reprezinte fiecare moneda
20 for(int nCount = 0; question — ⑤
    nCount < pDoc->GetCoinCount(); nCount++)
21 {
22     int y = 200 - 10 * nCount;
23     pDC->Ellipse(40, y, 100, y-30); — ⑥
24     pDC->Ellipse(40, y-10, 100, y-35);
25 }
26
27 // ** Refacem pensula curenta
28 pDC->SelectObject(pOldBrush); — ⑦
29 }

```

- ① Apelăm funcția `GetDocument()` a reprezentării pentru a obține un pointer la clasa document a aplicației.
- ② Obținem un pointer la pensula selectată curent în contextul dispozitiv `pDC`.
- ③ Creăm o pensulă plină care va desena cu culoarea galbenă.
- ④ Selectăm noua pensulă în contextul dispozitiv.
- ⑤ Iterăm până când au fost desenate toate monedele. Numărul acestora este obținut prin apelul funcției de acces oferită de document, `GetCoinCount()`.
- ⑥ Desenăm două elipse cu coordonatele deplasate pentru a reprezenta fiecare monedă.
- ⑦ Selectăm la loc pensula originală din contextul dispozitiv.

Primele două linii (liniile 4 și 5) ale funcției sunt generate automat de `AppWizard`. Funcția `GetDocument` întoarce un pointer la obiectul document, acesta fiind reținut în `pDoc`. Pointerul este folosit în bucla `for` din linia 20 pentru a obține numărul curent de monede, fiind apelată metoda de acces `GetCoinCount`.

În linia 9, în `pOldBrush` este reținut un pointer la pensula selectată curent în contextul dispozitiv, aceasta fiind refăcută la finalul funcției, în linia 28.

Liniile 12, 13 și 16 creează o pensulă plină de culoare galbenă în cadrul variabilei locale `br` și o selectează în contextul dispozitiv, pregătind desenarea monedei(-lor). Fiecare monedă este desenată prin două elipse, una deasupra alteia, obținându-se un efect tridimensional. În acest scop este apelată funcția `CDC::Ellipse`, așa cum puteți vedea în liniile 23 și 24.

VEDEȚI...

- Pentru informații privind contextele dispozitiv, vedeți pagina 323.
- Pentru informații privind trasarea graficii, vedeți pagina 372.

Utilizarea resurselor standard

Așa cum am menționat mai devreme în acest capitol, la proiect sunt adăugate automat mai multe resurse care să susțină elementele vizuale ale unei aplicații SDI. Acestea sunt meniul, bara cu instrumente, o pictogramă, o tabelă de acceleratori și șirul specific documentului.

Toate aceste resurse standard sunt complet funcționale. Spre exemplu, opțiunea **New** din meniul **File** va apela `CWinApp::OnFileNew`, ceea ce va duce la crearea unui nou document, același efect obținându-se și prin intermediul butonului **New** de pe bara cu instrumente. Aceasta nu înseamnă că nu puteți să modificați comportamentul implicit al acestor opțiuni. Puteți supradefini oricare dintre funcțiile de comandă, efectuând orice prelucrări doriți. Dacă opțiunile din meniu și de pe bara cu instrumente nu au sens pentru aplicație, le puteți elimina. Oricum, este mult mai uzual să lăsați opțiunile standard așa cum sunt și să adăugați propriile opțiuni, specifice aplicației. Aceasta se face foarte ușor, deoarece fiecare resursă poate fi editată prin intermediul editorului de resurse și atașată la funcția de prelucrare corespunzătoare cu `ClassWizard`.

Pentru exemplul `SDICoin` trebuie să adăugați două opțiuni de meniu, una care să adauge o monedă și cealaltă care să înlăture o monedă. La selectarea lor, aceste opțiuni vor apela fiecare câte o funcție a clasei document care va crește sau va scădea numărul de monede și va asigura actualizarea reprezentării. Pentru adăugarea opțiunilor de meniu urmați pașii de mai jos.

Adăugarea acceleratoarelor pentru opțiunile de meniu

La editarea proprietăților unei opțiuni de meniu puteți să specificați o combinație de taste alternativă, atașând-o la sfârșitul etichetei. Spre exemplu, eticheta `&ToolBarCtrl+T` a opțiunii de meniu corespunzătoare arată că apăsarea combinației `Ctrl+T` va fi echivalentă cu selectarea respectivei opțiuni. De asemenea, identificatorul `ID_VIEW_TOOLBAR` trebuie adăugat în tabela de acceleratori, definind aici combinația tastelor `T` și `Ctrl`.

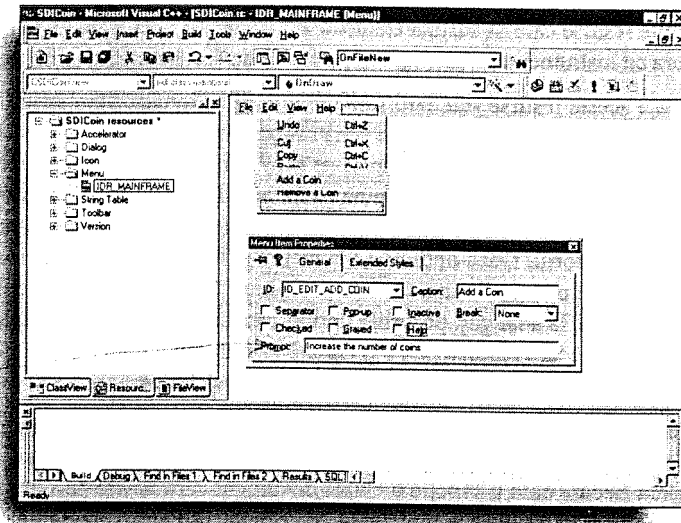
Adăugarea opțiunilor de meniu

1. În cadrul paginii `ResourceView`, expandați catalogul `Menu` și efectuați un dublu clic pe `IDR_MAINFRAME`. Resursa meniu va fi afișată în editorul de resurse.
2. Efectuați un clic pe elementul de meniu **Edit** afișat în editorul de resurse. Este deschis meniul derulant.
3. Efectuați un dublu clic pe elementul gol de la baza meniului derulant. Caseta de dialog `Menu Item Properties` este afișată ca în figura 12.7.
4. Specificați un identificator (**ID**) pentru elementul de meniu. Pentru cazul de față introduceți `ID_EDIT_ADD_COIN`.
5. Specificați o etichetă (**Caption**) pentru elementul de meniu. Pentru cazul de față introduceți `Add a Coin`.

6. Specificați o explicație (**Prompt**) pentru elementul de meniu. Pentru cazul de față introduceți `Increase the number of coins`.
7. Efectuați un dublu clic pe elementul gol de la baza meniului derulant.
8. Specificați un identificator pentru elementul de meniu. Pentru cazul de față introduceți `ID_EDIT_REMOVE_COIN`.

FIGURA 12.7

Caseta de dialog Menu Item Properties.



9. Specificați o etichetă pentru elementul de meniu. Pentru cazul de față introduceți `Remove a Coin`.
10. Specificați o explicație pentru elementul de meniu. Pentru cazul de față introduceți `Decrease the number of coins`.
11. Apăsați **Ctrl+W** pentru a apela **ClassWizard** sau selectați **ClassWizard** din meniul **View**.
12. Selectați pagina **Message Maps**.
13. Selectați `CSDICoinDoc` în caseta combinată **Class Name**.
14. Selectați `ID_EDIT_ADD_COIN` în caseta cu listă **Object IDs**.
15. Selectați **COMMAND** în caseta cu listă **Messages**, după care efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
16. Selectați `ID_EDIT_REMOVE_COIN` în caseta cu listă **Object IDs**.
17. Selectați **COMMAND** în caseta cu listă **Messages**, după care efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
18. Efectuați un clic pe butonul **Edit Code**.

Actualizarea reprezentării

După efectuarea modificărilor asupra datelor documentului, aproape întotdeauna este necesară reflectarea acestora în cadrul reprezentării. În acest scop sunt apelate funcții care duc la redesenarea reprezentărilor. În secțiunea anterioară ați adăugat două opțiuni de meniu și ați creat funcțiile de tratare corespunzătoare în cadrul clasei document. Acum editați cele două funcții așa cum o arată listingul 12.4.

LISTINGUL 12.4 LST12_2.CPP – Actualizarea reprezentării pe baza documentului

```

1 void CSDICoinDoc::OnEditAddCoin() — ❶
2 {
3     // TODO: Add your command code handler here
4
5     // ** Incrementam numarul de monede
6     m_nCoins++; — ❷
7
8     // ** Actualizam reprezentarea pentru a
9     // redesea teancul de monede
10    UpdateAllViews(NULL); — ❸
11 }
12
13 void CSDICoinDoc::OnEditRemoveCoin()
14 {
15     // TODO: Add your command code handler here
16
17     // ** Decrementam numarul de monede
18     if(m_nCoins > 0)
19         m_nCoins--;
20
21     // ** Actualizam reprezentarea pentru a redesea teancul de
22     // monede
23     UpdateAllViews(NULL);
24 }

```

❶ Funcția de tratare a unei comenzi de meniu, apelată la selectarea opțiunii de meniu **Add a Coin**.

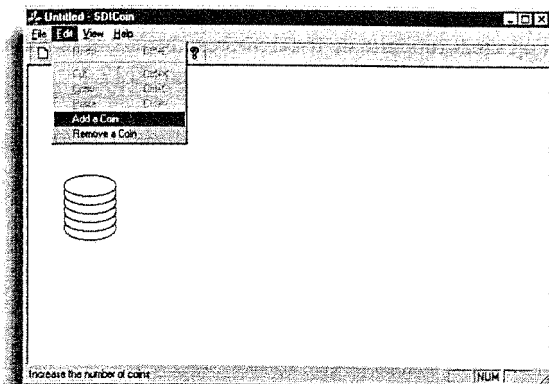
❷ Incrementăm variabila membru a documentului care reține numărul de monede.

❸ Determinăm apelarea funcțiilor `OnUpdate()` ale reprezentărilor asociate documentului, ceea ce va duce la redesenarea acestora.

Precum vedeți, fiecare dintre funcții incrementează sau decrementează `m_nCoins`, care constituie datele documentului. Apoi este apelată funcția `UpdateAllViews`, fiind transmis parametrul `NULL`. Această valoare `NULL` determină actualizarea tuturor reprezentărilor atașate la document. Actualizarea fiecărei reprezentări se face prin apelul funcției `OnUpdate` a acesteia. Efectul este invalidarea zonei client a ferestrei reprezentării și apelarea funcției `OnDraw`.

Acum asamblați și rulați proiectul SDICoin. Experimentați selectarea opțiunilor **Add a Coin** și **Remove a Coin**. Reprezentarea ar trebui să fie similară cu cea înfățișată în figura 12.8.

FIGURA 12.8
Aplicația SDICoin.



CAPITOLUL

13

Lucrul cu meniuri

Crearea și editarea resurselor de tip meniu

Scrierea funcțiilor de tratare pentru comenzile de meniu

Modul de implementare a meniurilor contextuale

Crearea și întreținerea dinamică a opțiunilor de meniu

Crearea și editarea resurselor de tip meniu

Meniurile reprezintă coloana vertebrală a interacțiunii dintre utilizatori și aplicațiile Windows, permițând navigarea rapidă în cadrul structurii de nivel înalt a interfeței cu utilizatorul. Meniurile conțin titluri de meniu (afișate în partea superioară, pe bara de meniuri) și elemente de meniu (opțiunile afișate la derularea meniurilor). Elementele de meniuri pot deschide la rândul lor submeniuri, oferind astfel căi ierarhice în codul aplicației, și pot fi dezactivate, validate sau tratate asemeni *butoanelor de opțiune*.

Meniurile contextuale vă permit să atașați obiectelor aplicației elemente de meniu care sunt afișate la cerere, prin efectuarea unui clic cu butonul drept al mouse-ului, oferind astfel utilizatorului un control mai rapid și mai clar asupra aplicației.

Resursele de tip meniu se creează și se întrețin lesne, prin intermediul editorului de resurse. Acesta vă permite să adăugați sau să înlăturați titluri de meniu și elemente de meniu într-o manieră WYSIWYG (What You See Is What You Get – ceea ce vezi este ceea ce ai).

VEDEȚI...

➤ Pentru mai multe informații privind editorul de resurse, vedeți pagina 45.

Adăugarea unor noi resurse de tip meniu

Meniurile sunt în mod normal afișate pe baza unei resurse de tip meniu (ca și machetele de dialog), care conține atât titlurile de meniu, cât și elementele de meniu. Editorul de resurse permite adăugarea unor noi resurse meniu. Oricum, dacă creați un proiect SDI sau MDI cu AppWizard veți avea la dispoziție un meniu inițial implicit, având identificatorul IDR_MAINFRAME. Dacă doriți să adăugați meniuri adiționale, operația de inserare a unei noi resurse meniu este descrisă prin secvența de pași următoare.

Meniuri create dinamic

Chiar dacă nu folosiți resurse meniu, puteți să creați meniuri dinamic și să adăugați opțiuni în aceste meniuri prin cod, așa cum veți vedea mai târziu în acest capitol.

Adăugarea unei noi resurse meniu

1. Selectați pagina ResourceView pentru a afișa resursele proiectului.
2. Efectuați un clic cu butonul drept pe elementul aflat la primul nivel și selectați opțiunea **Insert** din meniul afișat pentru a apela caseta de dialog Insert Resource.
3. Selectați **Menu** din lista cu tipurile de resurse noi.
4. Efectuați un clic pe butonul **New** pentru a insera noua resursă meniu. Sub nodul Menu ar trebui să apară noua machetă de meniu.

Utilizarea combinației de taste pentru inserarea unui meniu

Puteți sări peste pașii de la 1 la 4 folosind ca scurtătură combinația de taste Ctrl+2, care inserează o resursă meniu indiferent de unde vă aflați în Developer Studio. După apăsarea acestor taste ar trebui să fie afișată pagina de resurse, cu noul meniu selectat și afișat în fereastra de editare.

5. Efectuați un clic cu butonul drept pe resursa meniu și selectați opțiunea **Properties** din meniul contextual pentru a afișa proprietățile meniului.
6. Acum puteți înlocui identificatorul de resursă implicit cu un nume mai potrivit pentru meniu.
7. Ca alternativă, puteți să expandați resursele proiectului și să efectuați un clic cu butonul drept pe nodul **Menu**, ceea ce vă oferă posibilitatea de a selecta opțiunea **Insert Menu** pentru a adăuga o nouă resursă meniu, după care modificarea identificatorului implicit se face ca în pașii 5 și 6.

VEDEȚI...

- Pentru mai multe informații privind crearea aplicațiilor SDI, vedeți pagina 246.
- Pentru mai multe informații privind crearea aplicațiilor MDI, vedeți pagina 477.
- Pentru mai multe informații privind editorul de resurse, vedeți pagina 45.

Adăugarea titlurilor de meniu

Odată ce ați inserat o nouă resursă sau ați selectat una deja existentă, generată de infrastructură, puteți să editați resursa respectivă în cadrul ferestrei de editare. Meniul este afișat în cadrul editorului așa cum îl veți vedea la momentul rulării aplicației, cu titlurile de meniu aliniate de-a lungul marginii superioare a ferestrei de editare. Într-o resursă meniu nouă sau existentă puteți să inserați un nou titlu de meniu.

Inserarea unui nou titlu de meniu

1. Selectați în cadrul paginii ResourceView resursa meniu dorită, efectuând un dublu clic pe aceasta pentru a deschide meniul corespunzător în fereastra de editare.
2. Efectuați un clic pe dreptunghiul gol afișat în dreapta titlurilor de meniu existente pe bara de meniuri. Dreptunghiul va apărea selectat, în jurul lui fiind afișat un chenar alb.
3. Tastați numele noului titlu de meniu. Textul va apărea în interiorul dreptunghiului pe măsură ce tastați, fiind afișată și o casetă de dialog Menu Item Properties. De asemenea, sub noul titlu de meniu va apărea o casetă pentru un element de meniu, așa cum se vede în figura 13.1.
4. Selectați opțiunile necesare pentru titlul de meniu (aceste opțiuni sunt prezentate detaliat în secțiuni ulterioare ale acestui capitol, așa că nu ne vom ocupa acum de ele).
5. Apăsăți Enter pentru a încheia inserarea noului titlu de meniu.
6. Acum puteți să trageți și să plasați noul titlu de meniu în orice poziție de pe bara de meniu, modificând după dorință ordinea titlurilor de meniu.

Adăugarea elementelor de meniu

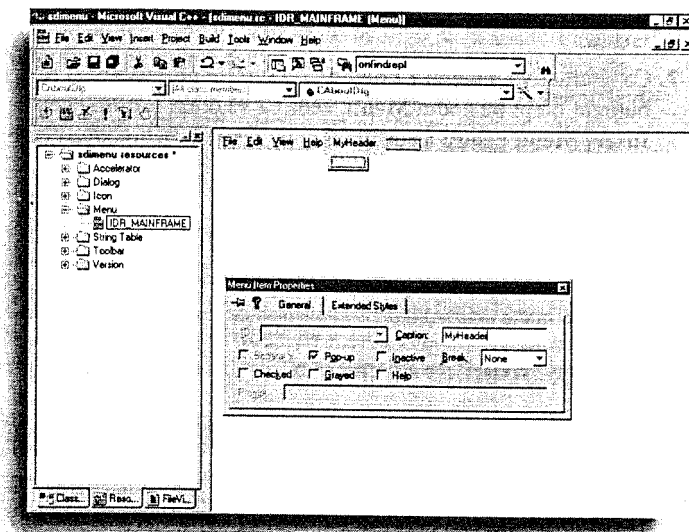
Inserarea unui nou element de meniu

1. Efectuați un clic pe titlul de meniu care va conține noul element de meniu.
2. Efectuați un clic pe dreptunghiul gol afișat sub elementele de meniu existente. În cazul în care adăugați un element de meniu în cadrul unui submeniu, efectuați clic pe titlul

submeniului pentru a afișa lista cu elementele conținute. Dreptunghiul va apărea selectat prin afișarea unui chenar alb în jurul său.

FIGURA 13.1

Inserarea unui nou titlu de meniu.



3. Tastați numele noului element de meniu. Textul va apărea în interiorul dreptunghiului pe măsură ce tastați, fiind afișată și o casetă de dialog Menu Item Properties (vedeți figura 13.2).
4. Selectați opțiunile necesare pentru elementul de meniu (aceste opțiuni sunt prezentate detaliat în secțiuni ulterioare ale acestui capitol, așa că nu ne vom ocupa acum de ele).
5. Efectuați un clic pe caseta **Prompt**. Aici puteți specifica textul care va apărea pe bara de stare la selectarea acestei opțiuni de meniu. Puteți preciza și un al doilea șir, aflat în continuare primul și separat de el prin caracterul linie nouă \n, acesta fiind afișat ca explicație pentru butonul asociat de pe bara cu instrumente. Un asemenea șir este ilustrat în caseta de editare **Prompt** din partea inferioară a figurii 13.2.
6. Efectuați un clic pe caseta de editare **ID**. Aici ar trebui să apară un identificator de meniu implicit, bazat pe elementul de meniu și pe titlul meniului care-l conține. Dacă este necesar, înlocuiți acest identificator implicit cu un altul mai potrivit pentru aplicație.
7. Acum puteți să trageți și să plasați noul element de meniu în orice poziție din cadrul meniului (chiar și ca un nou titlu de meniu).

Atribuirea de identificatori pentru meniuri

Identificatorul unui element de meniu este folosit pentru a mapa o funcție de tratare peste respectivul element, așa cum o descrie secțiunea „Adăugarea unei funcții de tratare pentru o comandă de meniu”, mai târziu în acest capitol. Dacă doriți să alegeți ca identificator

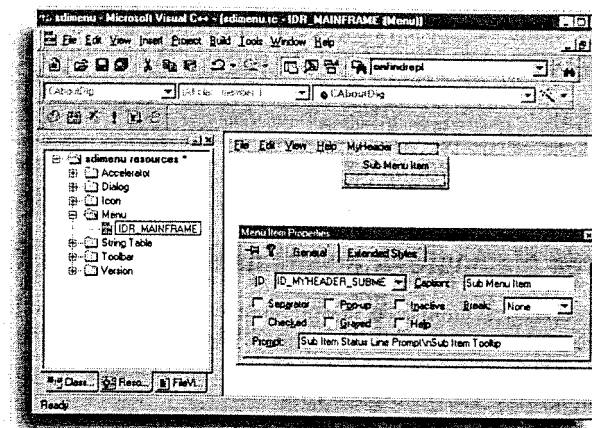
unul deja existent (așa cum ar fi cel al unui buton de pe bara cu instrumente), puteți să efectuați un clic pe butonul de derulare al casetei combinate și să selectați un identificator din lista afișată. Sunt cazuri în care ați putea dori să utilizați un același identificator pentru mai multe resurse meniu. Spre exemplu, ați putea avea două reprezentări diferite care afișează ambele un același meniu **File** prin care se apelează o funcție de tratare din clasa document. În acest caz, ați dubla meniul **File** și ați partaja identificatorii pentru a fi apelate aceleași funcții de tratare.

Utilizarea identificatorilor de meniu

Identificatorul asociat unui element de meniu este folosit ulterior pentru a vă informa că utilizatorul a selectat respectivul element. Windows trimite către fereastra părinte a meniului un mesaj WM_COMMAND corespunzător opțiunii selectate. Identificatorul în cauză însoțește mesajul. Acest identificator va fi verificat în cadrul hărții de mesaje a ferestrei de către macrodefiniția ON_COMMAND în scopul găsirii funcției de tratare corespunzătoare.

FIGURA 13.2

Inserarea unui nou element de meniu.



Până la adăugarea funcțiilor de tratare corespunzătoare, toate elementele de meniu noi rămân dezactivate în mod implicit.

Dacă precizați un șir informativ, așa cum se arată în pasul 5, respectiva valoare va fi înscrisă în tabela de șiruri cu același identificator ca cel al elementului de meniu. La compilarea aplicației, șirul informativ va fi afișat în linia de stare a aplicației.

Modificarea proprietăților unui element de meniu

Proprietățile unui element de meniu pot fi afișate efectuând un dublu clic pe elementul respectiv sau selectându-l și apăsând Alt+Enter. În caseta de dialog Properties veți putea modifica oricare dintre proprietățile elementului de meniu.

Poziția unui element de meniu poate fi modificată prin tragerea și plasarea acestuia într-o nouă poziție. Între celelalte elemente de meniu vor apărea bare de ghidare care vă arată unde va fi plasat elementul la momentul curent.

Un element de meniu poate fi șters prin selectarea acestuia și apăsarea tastei Delete. Dacă ștergeți un titlu de meniu, vor fi șterse toate elementele aflate în meniul respectiv (dar mai întâi vi se va solicita confirmarea).

VEDEȚI...

➤ Pentru amănunte privind tabelele de șiruri, vedeți pagina 45.

Adăugarea de separatori

Este posibilă adăugarea unor separatori care să grupeze elementele de meniu prin trasarea unor bare orizontale de-a latul casetei meniului (așa este cazul meniului **File** din Developer Studio). În acest scop, efectuați un dublu clic pe dreptunghiul gol care marchează un nou element pentru a afișa caseta de dialog Menu Item Properties; apoi efectuați un clic pe caseta de validare **Separator** pentru a selecta proprietatea de separator. La selectarea acestei proprietăți vor fi dezactivate toate controalele, mai puțin caseta de editare **Caption**, care va fi golită de conținut. Apăsați Enter pentru a închide caseta de dialog. Noul separator va apărea ca o bară orizontală. Acum puteți să-l trageți și să-l plasați în poziția dorită.

Transformarea separatorilor în elemente de meniu obișnuite

Puteți transforma un separator înapoi într-un element de meniu obișnuit tastând ceva în caseta de editare **Caption** din dialogul cu proprietățile acestuia.

Crearea elementelor de submeniu

Puteți să creați elemente de submeniu, acestea semănând cu titlurile de meniu prin aceea că afișează o listă cu elemente de meniu (vedeți figura 13.3). Pentru a crea un astfel de submeniu trebuie să adăugați un nou element de meniu și apoi să efectuați un clic pe caseta de validare **Pop-Up** din caseta de dialog Menu Item Properties, selectând în acest fel proprietatea de submeniu a elementului respectiv. În consecință, câmpul **ID** va fi voalat, iar în dreapta elementului de meniu va fi afișată o mică săgeată orientată spre dreapta, indicând un alt dreptunghi gol de inserare.

Acum puteți să inserați în acest submeniu noi elemente, exact așa cum ați insera elemente de meniu obișnuite. Aceste elemente de meniu pot fi la rândul lor submeniuuri, creând dacă este necesar un nou nivel de adâncime. Prin utilizarea submeniurilor este posibilă crearea unui sistem de meniuri bine ierarhizat.

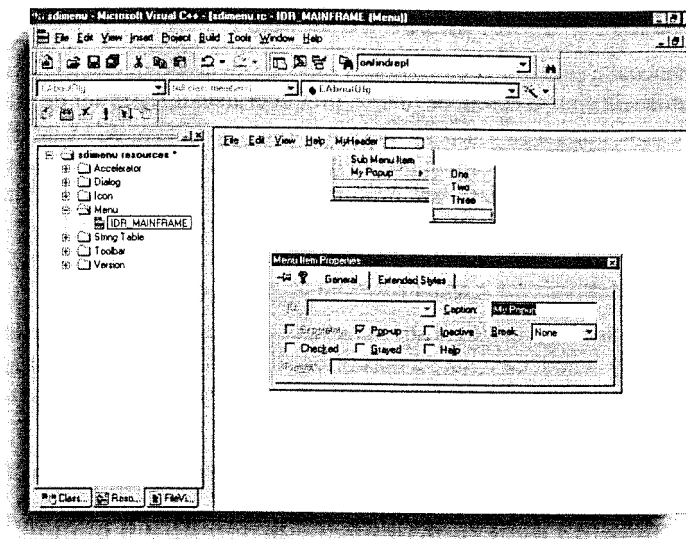
Adăugarea marcajelor de validare

Elementele de meniu pot fi însoțite de marcaje de validare care sunt afișate în partea stânga a acestora. Marcajele de validare pot fi manipulate prin cod și, prin intermediul editorului de resurse, pot fi afișate implicit alături de un element de meniu la apariția inițială a acestuia.

Pentru a determina afișarea implicită a unui marcaj de validare, efectuați un clic pe proprietatea **Checked** din caseta de dialog Menu Item Properties pentru fiecare element de meniu care doriți să fie afișat inițial împreună cu marcajul de validare.

FIGURA 13.3

Crearea elementelor de submeniu.



Specificarea tastelor de acces

Pentru elementele de meniu se pot defini taste de acces care să permită utilizatorului selectarea unui element prin apăsarea tastei de acces asociate. Aceste taste de acces sunt indicate prin sublinierea în cadrul unui element de meniu a caracterului care reprezintă tasta de acces pentru elementul respectiv.

Taste mnemonice

Aceste taste de acces poartă și numele de taste mnemonice, putând fi asociate controalelor dintr-o casetă de dialog pentru stabilirea focusului și accesarea unui control anume.

Pentru a specifica tasta de acces asociată unui element de meniu trebuie să inserați un caracter ampersand (&) înainte de caracterul corespunzător. La afișarea elementului de meniu, caracterul prefixat va apărea subliniat, iar tasta corespunzătoare va oferi accesul imediat la respectivul element.

Spre exemplu, dacă doriți ca tasta de acces pentru meniul **Acces Rapid** să fie R, va trebui să introduceți eticheta meniului sub forma **Acces &Rapid**, pe ecran fiind afișat **Acces Rapid**.

Dacă aveți nevoie chiar de un caracter ampersand, va trebui să introduceți două caractere ampersand consecutive. Astfel, **Un && Am&persand** va fi afișat sub forma **Un & Am&persand**.

VEDEȚI...

➤ Pentru mai multe amănunte privind tastele de acces, vedeți pagina 70.

Tratarea comenzilor de meniu

Comenzile provenite de la meniuri sunt tratate destul de asemănător cu cele generate de butoanele dintr-o casetă de dialog, prin intermediul funcțiilor de tratare a comenzilor. Pentru crearea unei astfel de funcții de tratare puteți apela la **ClassWizard**, acesta adăugând în harta de mesaje o intrare pentru mesajul `WM_COMMAND` provenit de la un element de meniu și având asociat identificatorul meniului respectiv. La selectarea de către utilizator a elementului de meniu, Windows va transmite un mesaj `WM_COMMAND` către acea fereastră a aplicației care conține meniul; în cazul în care în harta de mesaje există o intrare corespunzătoare, va fi apelată funcția de tratare asociată.

Tot prin intermediul intrărilor în harta de mesaje pe care le generează **ClassWizard** puteți adăuga și funcții de tratare a mesajelor provenite de la interfața cu utilizatorul, funcții care să controleze starea de activare/dezactivare a elementelor de meniu.

VEDEȚI...

➤ Pentru explicații amănunțite privind hărțile de mesaje, vedeți pagina 75.

Adăugarea unei funcții de tratare pentru o comandă de meniu

Odată creat un nou meniu, există posibilitatea de a adăuga o funcție de tratare care să fie apelată la selectarea acestuia, executându-se astfel codul pe care l-ați asociat cu respectivul element de meniu. Funcția de tratare poate fi implementată în orice clasă, obiectele acesteia primind mesajul de selecție a comenzii de meniu în momentul în care elementul de meniu este selectat. De regulă se alege fie clasa `document`, fie clasa reprezentării care implementează opțiunea de meniu. La fel de bine, însă, puteți să răspundeți la comanda de meniu în clasa aplicației sau a ferestrei cadru. Pentru a alege cât mai bine clasa de implementare a funcției de tratare, luați în considerație accesul la date și metodele diferitelor obiecte ale aplicației care sunt necesare. Pentru adăugarea funcției de tratare puteți folosi **ClassWizard**.

Adăugarea unei funcții de tratare pentru o comandă de meniu cu **ClassWizard**

1. Selectați elementul de meniu pentru care vreți să adăugați o funcție de tratare.
2. Apăsați **Ctrl+W** sau efectuați un clic pe meniul **View** și selectați **ClassWizard** pentru a apela **ClassWizard**. În lista **Object IDs** ar trebui să fie evidențiat identificatorul elementului de meniu selectat, așa cum o arată figura 13.4.
3. Casetă combinată **Class Name** înfățișează clasa destinație pentru noua funcție de tratare. Efectuați un clic pe butonul de derulare al casetei combinate dacă doriți să vedeți toate clasele în care poate fi implementată noua funcție de tratare a meniului. Puteți, așadar, să selectați clasa ce va conține implementarea, aceasta fiind de regulă fie reprezentarea care implementează opțiunea de meniu, fie o clasă `document`, în funcție de relevanța acestora.
4. Efectuați un dublu clic pe mesajul `COMMAND` din caseta cu listă **Messages** pentru a apela caseta de dialog **Add Member Function** (vedeți figura 13.5). Ca alternativă, puteți să selectați mesajul `COMMAND` și să efectuați un clic pe butonul **Add Function**.

FIGURA 13.4

ClassWizard afișează identificatorul elementului de meniu selectat.

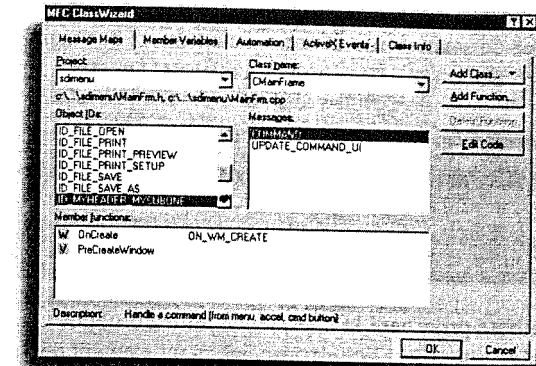
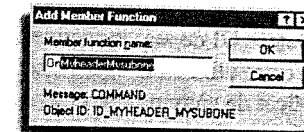


FIGURA 13.5

Casetă de dialog **Add Member Function** va permite să modificați numele noii funcții de tratare.



5. Numele implicit al noii funcții de tratare se bazează pe identificatorul elementului de meniu. Înainte de crearea funcției aveți posibilitatea de a modifica acest nume implicit, prin intermediul casetei de editare **Member Function Name**.
6. Efectuați un clic pe **OK** pentru a adăuga noua funcție membru.
7. Noua funcție membru ar trebui să apară în lista **Member Functions** din partea inferioară a casetei de dialog **ClassWizard**.
8. Acum puteți edita codul noii funcții membru, efectuând un clic pe butonul **Edit Code**.

Odată creată noua funcție de tratare, puteți adăuga codul de implementare specific aplicației. De exemplu, funcția de tratare de mai jos (implementată în clasa `document`) se rezumă la afișarea unei casete de mesaj care arată că elementul de meniu a fost selectat:

```
void CSdimenuDoc::OnMyheaderMysubone()
{
    AfxMessageBox("You picked the 1st menu item");
}
```

VEDEȚI...

- Pentru a înțelege transmiterea mesajelor în cadrul unei aplicații **SDI**, vedeți pagina 250.
- Pentru explicații detaliate privind hărțile de mesaje, vedeți pagina 75.

Adăugarea unei funcții de tratare a mesajelor provenite de la interfața cu utilizatorul

Funcțiile care tratează interfața cu utilizatorul sunt responsabile de aspectul, stilul și comportamentul fiecărui element de meniu. Înainte de afișarea unui meniu sunt apelate

funcțiile corespunzătoare de tratare a interfeței cu utilizatorul, aplicația având astfel posibilitatea de a activa sau dezactiva elementele de meniu, de a adăuga sau înlătura marcaje de validare sau de a modifica etichetele opțiunilor.

Când sunt apelate funcțiile de tratare a interfeței cu utilizatorul?

Funcțiile de tratare a interfeței cu utilizatorul asociate meniurilor sunt apelate după ce utilizatorul efectuează clic pe un titlu de meniu, dar înainte de afișarea meniului. Acest sistem de actualizare la cerere face ca tratarea meniurilor să fie mai rapidă decât în cazul în care atributele acestora ar fi fost actualizate de fiecare dată când se modifică starea programului, chiar dacă utilizatorul nu ar fi putut să vadă elementele de meniu afectate.

Aceste atribute privitoare la interfața cu utilizatorul pot fi modificate prin apelul funcțiilor de acces ale unui obiect `CCmdUI` asociat cu un element de meniu. La orice apelare a unei funcții de tratare a interfeței cu utilizatorul (UI – user interface), acestea îi va fi transmis un pointer la un obiect `CCmdUI`.

De pildă, funcția de tratare UI de mai jos activează un element de meniu pentru a permite selectarea acestuia de către utilizator. În acest scop este apelată funcția `Enable()` a obiectului `CCmdUI` indicat de către pointerul `pCmdUI`:

```
void CSdimenuDoc::OnUpdateMyheaderMysubone(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(TRUE);
}
```

Secțiunile următoare ale acestui capitol prezintă mai pe larg aspectele legate de interfața cu utilizatorul.

Pentru adăugarea în aplicație a unei funcții de tratare UI, puteți să urmați secvența de pași pentru adăugarea unei funcții de tratare a unei comenzi de meniu prin intermediul `ClassWizard`, prezentată mai devreme în capitolul de față, cu deosebirea că din lista **Messages** nu veți mai selecta în pasul 4 mesajul `COMMAND`, ci mesajul `UPDATE_COMMAND_UI`.

Activarea și dezactivarea opțiunilor de meniu

Opțiunile de meniu pot fi activate sau dezactivate apelând funcția `Enable()` a obiectului `CCmdUI`. Dacă transmiteți valoarea `TRUE` la apelul funcției, meniul va fi activat; în caz contrar, opțiunea de meniu va fi dezactivată și afișată voalat.

Ați putea să rețineți starea de activare prin intermediul unei variabile membru a clasei, având tip boolean, transmitând-o apoi funcției `Enable()` a obiectului `CCmdUI` în cadrul funcției de tratare UI, ca mai jos:

```
pCmdUI->Enable(m_bMySubItemEnableStatus);
```

Reținând starea de activare în variabila `m_bMySubItemEnableStatus` a aplicației, elementul de meniu va fi activat sau dezactivat în mod corespunzător.

Afișarea sau înlăturarea unui marcaj de validare

Este posibilă afișarea unui marcaj de validare (sau *bifă*) alături de un element de meniu cu scopul de a indica utilizatorului o valoare de adevăr relevantă pentru aplicație. Prezența marcajului de validare poate fi stabilită prin apelul funcției `SetCheck()` a obiectului `CCmdUI`, transmitând valoarea zero pentru înlăturarea marcajului sau valoarea unu pentru afișarea acestuia.

Dacă doriți să implementați un comutator simplu prin intermediul unui element de meniu, permițând utilizatorului să comute afișarea marcajului de validare prin selectarea elementului respectiv, ați putea opta pentru o implementare ca cea prezentată în listingul 13.1.

LISTINGUL 13.1 LST14_1.CPP – Implementarea unui element de meniu comutator cu ajutorul marcajelor de validare

```
1 void CSdimenuDoc::OnMyheaderMysubone()
2 {
3     // ** Comutam starea curenta
4     m_nToggleState = m_nToggleState==0 ? 1 : 0; — ❶
5 }
6
7 void CSdimenuDoc::OnUpdateMyheaderMysubone
8     (CCmdUI* pCmdUI)
9 {
10     // ** Activam elementul de meniu
11     pCmdUI->Enable(TRUE);
12
13     // ** Stabilim starea de comutare
14     pCmdUI->SetCheck(m_nToggleState); — ❷
15 }
```

- ❶ Ca răspuns la selectarea elementului de meniu de către utilizator, comutăm valoarea `m_nToggleState` între 0 și 1.
- ❷ Dacă `m_nToggleState` este 1 atunci afișăm un marcaj de validare, iar dacă este 0 atunci înlăturăm marcajul de validare.

În listingul 13.1, funcțiile de tratare a comenzii și a actualizării comenzii sunt folosite împreună pentru a implementa un comutator. Funcția de tratare a comenzii, `OnMyheaderSubone()`, implementată între liniile 1 și 5, întreține starea comutatorului în variabila membru `m_nToggleState`. Această variabilă membru ar putea fi definită în cadrul clasei ca un întreg:

```
int m_nToggleState;
```

În linia 4 se folosește operatorul condițional `?` pentru a verifica dacă valoarea curentă `m_nToggleState` este zero. Dacă da, `m_nToggleState` devine unu; altfel, variabila devine zero, realizându-se astfel comutarea stării curente.

Funcția de tratare a interfeței cu utilizatorul este `OnUpdateMyheaderMysubone()`, implementată între liniile 7 și 14. Linia 10 asigură faptul că elementul de meniu este întotdeauna activat, apelând funcția `Enable()` a obiectului `CCmdUI`.

Starea curentă de comutare este reflectată de elementul de meniu prin afișarea sau înlăturarea unui marcaj de validare, în linia 13 fiind apelată funcția `SetCheck()` a obiectului `CCmdUI`. `SetCheck()` primește ca parametru variabila `m_nToggleState`, afișând sau înlăturând marcajul de validare în funcție de valoarea `m_nToggleState`.

Modificarea dinamică a etichetelor de meniu

Funcțiile de tratare UI pot fi utilizate și pentru modificarea etichetei unui element de meniu, transmițând o nouă etichetă funcției `SetText()` a obiectului `CCmdUI`. Eticheta meniului va fi înlocuită de noul șir de caractere, fiind luate în considerare și eventualele taste de acces specificate prin intermediul simbolului ampersand (&).

Transmiterea mai departe a mesajelor pentru actualizarea interfeței cu utilizatorul

Puteți să adăugați în două clase diferite intrări în harta de mesaje care să corespundă unui același mesaj de actualizare a interfeței (în clasele document și reprezentare, de pildă). În mod normal, însă, prima funcție de tratare va împiedica transmiterea mai departe a mesajului, către a doua funcție. Dacă doriți ca mesajul să treacă în unele cazuri de prima funcție pentru a fi tratat de a doua, ați putea să implementați un sistem de tratare pe două niveluri. În acest scop va trebui să apelați funcția `ContinueRouting()` a obiectului `CCmdUI` pentru ca mesajul să meargă mai departe și să ajungă la următoarea funcție de tratare din lanț.

Ați putea să modificați eticheta elementului de meniu din exemplul de comutator de mai devreme, astfel încât să afișeze **On** sau **Off** în funcție de starea de comutare, adăugând la sfârșitul funcției de tratare UI următoarea linie de cod:

```
pCmdUI->SetText(m_nToggleState?"&On":"&Off");
```

În linia de mai sus, operatorul condițional inline `?` este utilizat pentru a transmite unul din două șiruri de caractere, în funcție de valoarea `m_nToggleState`.

Implementarea meniurilor de context

Un *meniu de context* este un meniu care poate fi afișat oriunde pe ecran (în principiu) în momentul în care utilizatorul efectuează un clic cu butonul drept pe un obiect al aplicației. Utilizatorul poate apoi să selecteze din meniu opțiuni specifice respectivului obiect.

Pentru afișarea unui meniu de context puteți să adăugați o nouă resursă meniu. Aceasta ar trebui creată cu ajutorul editorului de resurse, așa cum am descris pas cu pas mai devreme, și editată așa cum o arată secțiunea „Crearea și editarea resurselor de tip meniu”.

Implementarea în aplicații a meniurilor de context se poate face prin crearea unei funcții de tratare a mesajului `WM_CONTEXTMENU` din Windows, acesta fiind transmis unei aplicații în momentul în care se efectuează un clic cu butonul drept al mouse-ului undeva în fereastra aplicației. În cadrul funcției de tratare veți încărca resursa meniu corespunzătoare și veți deschide meniul de context prin apelul funcției

`TrackPopupMenu()`. Pentru implementarea funcționalității elementelor de meniu puteți adăuga funcții obișnuite de tratare a comenzilor de meniu.

Deschiderea unui meniu de context

Adăugarea unei funcții de tratare a mesajului `WM_CONTEXTMENU` pentru apelarea unui meniu de context se poate face urmând pașii de mai jos.

Mesajul WM_CONTEXTMENU

Mesajul `WM_CONTEXTMENU` este generat de către procedura de fereastră implicită la primirea mesajului `WM_RBUTTONDOWN`. În cazul în care interceptați mesajul `WM_RBUTTONDOWN`, dar nu apelați funcția de tratare din clasa de bază, aplicația nu va primi niciodată mesajul `WM_CONTEXTMENU`.

Adăugarea unei funcții de tratare pentru apelarea unui meniu de context

1. Selectați pagina **ClassView** a ferestrei spațiului de lucru al proiectului și, dacă este nevoie, efectuați un clic pe semnul plus din partea superioară pentru a afișa clasele componente ale proiectului.
2. Efectuați un clic cu butonul drept pe clasa reprezentare în care doriți să implementați noul meniu de context și selectați din meniul afișat opțiunea **Add Windows Message Handler** pentru a apela caseta de dialog **New Windows Message and Event handlers**.
3. Selectați mesajul `WM_CONTEXTMENU` din lista **New Windows Messages/Events** și efectuați un clic pe butonul **Add and Edit** pentru a adăuga o funcție de tratare a mesajului privind apelarea meniului de context.
4. Acum ar trebui să apară o nouă funcție de tratare generată de **ClassWizard** și având numele implicit `OnContextMenu()`. În continuare puteți să adăugați propriul cod la implementarea standard.

Odată inserată noua funcție de tratare, puteți să implementați codul pentru afișarea unui meniu de context. Ați putea dori să afișați meniul contextual numai în cazul în care cursorul mouse-ului se află deasupra unui obiect anume, sau ați putea să afișați meniuri de context distincte în funcție de obiectul selectat. În acest scop puteți să folosiți parametrul `point` de tip `CPoint` pentru a determina care este obiectul aplicației sau zona din fereastră unde s-a efectuat clic, afișând condiționat meniul de context (sau afișând un anume meniu de context). Metodele specifice sunt prezentate în capitolul 8, „Tratarea evenimentelor generate de mouse”.

Pentru afișarea meniului de context trebuie să declarați mai întâi un obiect `CMenu` care să implementeze respectivul meniu (clasa `CMenu` este o clasă MFC de încapsulare care permite întreținerea și accesarea unui obiect `HMENU` din Windows). Noul obiect `CMenu` poate fi apoi inițializat cu resursa meniu corespunzătoare, apelând funcția `LoadMenu()` a acestuia și transmițând identificatorul resursei meniu. În consecință, obiectul `CMenu` va fi inițializat cu titlul de meniu al resursei. Dar pentru afișarea meniului de context veți avea nevoie de elementele subordonate respectivului titlu de meniu. În acest scop veți apela funcția `GetSubMenu()`, transmițând valoarea zero pentru a solicita primul element subordonat.

Crearea dinamică a meniurilor de context

În loc să încărcați un meniu de context predefinit pe baza unei resurse meniu, ați putea dori să afișați un meniu al cărui conținut să depindă de obiectul anume pe care utilizatorul a efectuat clic (și nu de tipul obiectului). Într-o astfel de situație ar putea fi de preferat să apelați funcția `CreatePopupMenu()` a clasei `CMenu` pentru a crea meniul, după care ați apela la `AppendMenu()` pentru a adăuga elemente de meniu în noul meniu. Mai departe veți apela la fel `TrackPopupMenu()` pentru a afișa meniul. Mai multe amănunte despre funcția `AppendMenu()` pot fi găsite în secțiunea „Adăugarea dinamică a elementelor de meniu”, mai târziu în acest capitol.

Pe baza meniului obținut puteți apela apoi funcția `TrackPopupMenu()` care afișează și controlează meniul de context. `TrackPopupMenu()` așteaptă un număr de patru parametri. Primul este un indicator de aliniere care specifică unde va fi afișat meniul relativ la poziția precizată. Dacă doriți ca mouse-ul să se afle în stânga meniului contextual, transmiteți valoarea `TPM_LEFTALIGN` ca prim parametru, împreună cu poziția cursorului pe post de poziție de afișare a meniului. Ceilalți indicatori de aliniere disponibili sunt prezentați în tabelul 13.1. Al doilea și al treilea parametru precizează coordonatele pe orizontală și pe verticală ale poziției de afișare a meniului. Aici veți transmite de regulă valorile `point.x` și `point.y` transmise funcției `OnContextMenu()`, astfel încât meniul contextual să fie afișat relativ la poziția cursorului de mouse. Ultimul parametru este un pointer la obiectul fereastră, transmițându-se de obicei operatorul `this` din C++ pe post de pointer la obiectul reprezentare curent.

TABELUL 13.1 Indicatori de aliniere pentru `TrackPopupMenu()`

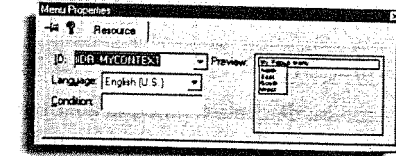
Numele indicatorului	Descriere
<code>TPM_LEFTALIGN</code>	Aliniază meniul astfel încât poziția specificată să se afle în partea stângă
<code>TPM_RIGHTALIGN</code>	Aliniază meniul astfel încât poziția specificată să se afle în partea dreaptă
<code>TPM_TOPALIGN</code>	Aliniază meniul astfel încât poziția specificată să se afle deasupra meniului
<code>TPM_BOTTOMALIGN</code>	Aliniază meniul astfel încât poziția specificată să se afle dedesubtul meniului
<code>TPM_CENTERALIGN</code>	Aliniază meniul astfel încât poziția specificată să fie centrată pe orizontală
<code>TPM_VCENTERALIGN</code>	Aliniază meniul astfel încât poziția specificată să fie centrată pe verticală
<code>TPM_HORIZONTAL</code>	În cazul în care nu există spațiu suficient pe ecran, alinierea orizontală primează
<code>TPM_VERTICAL</code>	În cazul în care nu există spațiu suficient pe ecran, alinierea verticală primează

La efectuarea apelului, meniul de context va fi afișat și utilizatorul va putea să selecteze una dintre opțiunile acestuia. Dacă este selectat un element sau dacă utilizatorul efectuează un clic altundeva, meniul de context dispare.

Ca exemplu, ați putea să implementați un meniu de context care permite utilizatorului să aleagă între opțiunile North, East, South și West, pornind de la o aplicație SDI generată cu AppWizard. Mai întâi ar trebui să adăugați o nouă resursă meniu corespunzătoare meniului de context, așa cum se vede în figura 13.6.

FIGURA 13.6

Inserarea unei noi resurse meniu pentru crearea unui meniu de context.



Odată inserată noua resursă meniu, puteți adăuga o funcție de tratare pentru mesajul `WM_CONTEXTMENU` urmând pașii din secvența „Adăugarea unei funcții de tratare pentru apelarea unui meniu de context”, prezentată mai devreme în acest capitol. Adăugați funcția de tratare în cadrul clasei reprezentare a aplicației SDI și implementați codul care se ocupă de încărcarea și controlarea meniului de context, așa cum o arată listingul 13.2.

LISTINGUL 13.2 LST14_2.CPP – Funcția `OnContextMenu()` de tratare a mesajului `WM_CONTEXTMENU` încarcă și controlează meniul de context

```

1 void CSDimenuView::OnContextMenu(CWnd* pWnd,
   CPoint point)
2 {
3     // ** Declaram un obiect CMenu
4     CMenu menuPopup;
5
6     // ** Il initializam cu resursa meniu de context
7     menuPopup.LoadMenu(IDR_MYCONTEXT);
8
9     // ** Afișăm și controlăm noul meniu
10    menuPopup.GetSubMenu(0)->TrackPopupMenu(
11        TPM_LEFTALIGN, point.x, point.y, this);
12 }
```

❶ Această linie afișează meniul de context și urmărește selecția efectuată de utilizator.

Linia 4 din listingul 13.2 declară un nou obiect `CMenu` sub forma variabilei `menuPopup`. Acesta este apoi inițializat în linia 7 cu elementele de meniu din resursa meniu `IDR_MYCONTEXT`.

În linia 10 este apelată `TrackPopupMenu()`, specificându-se că alinierea se face considerând punctul de coordonate aflat în stânga și transmițând poziția cursorului care a fost primită de funcția de tratare a meniului de context. Este transmis, de asemenea, un pointer la obiectul reprezentare curent de tip `CSDimenuView` prin intermediul cuvântului cheie `this` din C++.

Pentru a afișa meniul adecvat, `TrackPopupMenu()` trebuie apelată pe baza meniului inclus în meniul principal al resursei. Un pointer la acest meniu este întors de către apelul de funcție `menuPopup.GetSubMenu(0)` de la începutul liniei 10.

VEDEȚI...

► Pentru mai multe detalii privind testul de clic, vedeți pagina 180.

Tratarea comenzilor din meniul de context

După adăugarea meniului de context puteți să adăugați funcții de tratare asociate elementelor din acest meniu, exact așa cum ați proceda în cazul unui meniu obișnuit (aspecte prezentate în secțiunea „Tratarea comenzilor de meniu”, mai devreme în acest capitol).

Caseta de dialog Adding a Class din ClassWizard

După inserarea noii resurse meniu, ClassWizard afișează la prima apelare caseta de dialog Adding a Class pentru că a descoperit noua resursă. Nu este nevoie să creați sau să selectați o clasă de încapsulare, așa că puteți ignora această casetă de dialog efectuând un clic pe butonul Cancel.

Puteți să folosiți ClassWizard pentru a adăuga funcții de tratare a comenzilor de și a interfeței cu utilizatorul pentru fiecare opțiune de meniu, asemeni funcțiilor de mai jos asociate opțiunii **North**:

```
void CSdimenuView::OnMypopupmenuNorth()
{
    AfxMessageBox("North");
}
```

Funcția de tratare a comenzii de meniu afișează într-o casetă de mesaj cuvântul North atunci când este selectată opțiunea **North** a meniului.

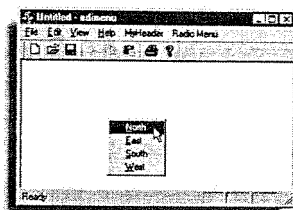
```
void CSdimenuView::OnUpdateMypopupmenuNorth(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(TRUE);
}
```

Funcția de tratare a interfeței cu utilizatorul activează opțiunea de meniu, apelând funcția `Enable()` pe baza pointerului la obiectul `CCmdUI` transmis funcției.

Puteți să operați toate aceste modificări într-o aplicație SDI standard și apoi să asamblați și să rulați aplicația. Efectuarea unui clic cu butonul drept oriunde în fereastra reprezentării va duce la afișarea noului meniu de context, așa cum o ilustrează figura 13.7.

FIGURA 13.7

Meniul contextual afișat la efectuarea unui clic cu butonul drept al mouse-ului.



VEDEȚI...

► Pentru mai multe informații privind tratarea evenimentelor generate de clicurile de mouse, vedeți pagina 169.

Crearea și accesarea obiectelor meniu

MFC pune la dispoziție o clasă `CMenu` care încapsulează un indicator de meniu (`HMENU`) și simplifică operațiile asupra acestuia. Puteți să folosiți `CMenu` pentru a genera prin cod propriile meniuri (în loc să le încărcați), având și posibilitatea de a adăuga și elimina dinamic elemente de meniu ale meniurilor existente. `CMenu` oferă o mulțime de funcții membru care acoperă fiecare aspect al afișării și manipulării meniurilor. Secțiunile care urmează descriu câteva dintre cele mai importante funcții din `CMenu`.

Accesarea indicatorului de meniu

Indicatorul de meniu care se ascunde în spatele obiectului `CMenu` poate fi accesat prin intermediul variabilei membru `m_hMenu`. Acest lucru s-ar putea dovedi util în cazul în care trebuie să folosiți funcții Win32 în locul echivalențelor MFC.

Inițializarea obiectului CMenu

Înainte de a putea utiliza un obiect `CMenu` pe care l-ați declarat, trebuie să-l inițializați încărcând un meniu dintr-o resursă, creând un nou meniu sau atașând obiectul meniu la un meniu deja existent din Windows.

Fiecare dintre diversele funcții de inițializare întoarce `TRUE` în cazul în care funcția reușește și meniul este inițializat sau, respectiv, `FALSE` dacă meniul nu a fost inițializat:

■ Încărcarea unui meniu dintr-o resursă

Puteți să încărcați un meniu prin intermediul funcției membru `LoadMenu()`, transmitând identificatorul de resursă al meniului în cauză, așa cum o ilustrează listingul 13.2 din secțiunea anterioară. În consecință, meniul va fi încărcat din resursă și va fi utilizat pentru crearea elementelor de meniu corespunzătoare.

■ Crearea de noi meniuri

`CreateMenu()` va crea dinamic o resursă meniu goală, fără nici un element de meniu. În continuare puteți să adăugați elemente de meniu care să formeze noul meniu, așa cum o arată secțiunea următoare.

Ca alternativă, puteți apela `CreatePopupMenu()` pentru a crea un meniu derulant care va putea fi utilizat ca submeniu pentru un element sau ca meniu de context.

■ Atașarea la un meniu existent

Puteți apela `Attach()` pentru a atașa la obiectul `CMenu` un meniu existent din Windows, transmitând în apel indicatorul `HMENU` către meniul în cauză. Acest indicator poate fi obținut pe baza unui obiect `CWnd` (sau derivat din `CView`) asociat, prin apelul funcției `GetMenu()` a respectivului obiect fereastră. Odată ce ați încheiat manipularea meniului, puteți să-l detașați de obiectul `CMenu` apelând `Detach()`.

După inițializarea obiectului CMenu, puteți să adăugați, să înlăturați sau să modificați elemente de meniu pentru a îndeplini cerințele aplicației, așa cum o arată secțiunile următoare.

VEDEȚI...

► Pentru mai multe informații despre ferestre și reprezentări și despre clasele CWnd și CView, vedeți pagina 250.

Adăugarea dinamică a elementelor de meniu

Adăugarea într-un meniu a unui nou element de meniu se poate face apelând `AppendMenu()` sau `InsertMenu()`, după cum elementul de meniu trebuie adăugat la sfârșitul meniului sau inserat în meniu într-o anumită poziție.

Funcția `AppendMenu()` așteaptă trei parametri. Primul parametru necesar este un indicator care specifică tipul noului element de meniu și o combinație de valori de stil. De obicei veți transmite `MF_STRING` ca prim parametru, specificând astfel că noul element conține un pointer la un șir obișnuit de caractere, terminat cu caracterul nul (puteți să folosiți și obiecte `CString`). Ca alternativă, puteți crea un element de tip separator transmițând valoarea `MF_SEPARATOR`. Aceste tipuri de bază pot fi combinate apoi cu ceilalți indicatori prezentați în tabelul 13.2, adăugând valorile acestora prin intermediul operatorului de adunare (+) sau al celui de SAU logic (|).

Prin al doilea parametru se poate transmite un identificator pentru mesajele de comandă sau un indicator `HMENU` al unui meniu derulant, în cazul în care a fost specificat indicatorul `MF_POPUP`.

Prin al treilea parametru puteți transmite un șir de caractere care va reprezenta eticheta elementului de meniu (sau puteți lăsa valoarea `NULL` implicită în cazul elementelor de tip separator).

Verificarea unicității pentru indicatorii de meniuri

Este bine să vă asigurați că indicatorii de meniuri sunt unici și nu există conflicte între aceștia. Această verificare se poate face deschizând fișierul `resource.h` din proiect (afișat în pagina `FileView`) și căutând noul identificator. În cazul în care acesta nu este găsit în `resource.h` și nu este folosit în vreun meniu dinamic, utilizarea sa nu ar trebui să ridice probleme.

TABELUL 13.2 Indicatori pentru meniuri utilizați de către funcțiile `AppendMenu()` și `InsertMenu()`

Indicator	Descriere
<code>MF_CHECKED</code>	Stabilește indicatorul pentru marcaj de validare, având ca efect afișarea unui marcaj de validare alături de elementul de meniu
<code>MF_UNCHECKED</code>	Anulează indicatorul pentru marcaj de validare al meniului asociat
<code>MF_ENABLED</code>	Activează elementul de meniu
<code>MF_DISABLED</code>	Dezactivează elementul de meniu
<code>MF_GRAYED</code>	Identice cu <code>MF_DISABLED</code>

Indicator	Descriere
<code>MF_POPUP</code>	Elementul de meniu are asociat un submeniu specificat de către identificatorul transmis ca parametru
<code>MF_MENUBARBREAK</code>	Elementul de meniu va fi plasat pe o nouă linie (sau pe o nouă coloană, în cazul meniurilor derulante, fiind separat printr-o linie verticală)
<code>MF_MENUBREAK</code>	Similar cu <code>MF_MENUBARBREAK</code> , dar fără separarea prin linie verticală

Funcția alternativă `InsertMenu()` vă permite să inserați noul element de meniu într-o poziție precisă sau înaintea unui element de meniu cu un identificator bine precizat. Dacă specificați poziția prin index, transmiteți ca prim parametru respectivul index pornind de la zero și adăugați indicatorul `MF_BYPOSITION` printre indicatorii combinați în cadrul celui de-al doilea parametru.

Pentru a insera noul element de meniu înaintea unui element existent având un identificator cunoscut, transmiteți acest identificator ca prim parametru și includeți `MF_BYCOMMAND` printre indicatorii care formează al doilea parametru.

Ceilalți trei parametri sunt identici cu cei transmiși funcției `AppendMenu()`. Spre exemplu, în loc să încărcați un meniu de context (așa cum o arată listingul 13.2 din secțiunea anterioară), ați putea să creați un meniu derulant și să atașați elementele de meniu dorite, așa cum o ilustrează listingul 13.3. Această abordare vă oferă un control mai bun asupra elementelor de meniu atașate și vă permite să adăugați în mod dinamic elemente de meniu specifice aplicației. Listingul 13.3 prezintă, de asemenea, diferitele moduri în care pot fi adăugate noile elemente de meniu, precum și efectele diferitelor combinații de indicatori.

LISTINGUL 13.3 LST14_3.CPP – Crearea dinamică și afișarea unui meniu de context

```

1 #define ID_MENU_RED      5001
2 #define ID_MENU_GREEN    5002
3 #define ID_MENU_BLUE     5003
4 #define ID_MENU_YELLOW   5004
5
6 void CSdimenuView::OnContextMenu(CWnd* pWnd,
7                                     CPoint point)
8 {
9     // Declaram un obiect CMenu
10    CMenu menuPopup;
11
12    // Cream resursa meniu
13    if (menuPopup.CreatePopupMenu())
14    {
15        // Adaugam elemente de meniu simple

```

(continuare)

LISTINGUL 13.3 Continuare

```

15     menuPopup.AppendMenu(MF_STRING, ID_MENU_RED, "&Red");
16
17     // Inseram un element la inceputul meniului
18     menuPopup.InsertMenu(0, MF_BYPOSITION | MF_STRING, — ❸
19                         ID_MENU_GREEN, "&Green");
20
21
22     menuPopup.AppendMenu(MF_SEPARATOR);
23
24     // Adaugam un element validat
25     menuPopup.AppendMenu(MF_STRING | MF_CHECKED,
26                         ID_MENU_BLUE, "&Blue");
27
28     // MF_MENUBARBREAK
29     menuPopup.AppendMenu(MF_STRING | MF_MENUBARBREAK, — ❹
30                         ID_MENU_YELLOW, "&Yellow");
31     menuPopup.TrackPopupMenu(TPM_LEFTALIGN,
32                             point.x, point.y, this);
33 }
34 }

```

- ❶ Identificatorii de meniu unici sunt definiți aici.
- ❷ Dacă crearea meniului derulant reușește, elementele de meniu pot fi adăugate în siguranță.
- ❸ Observați că acest element este inserat prin poziție, cu indicele zero (deasupra primului element).
- ❹ Indicatorul pentru separator de bară de meniuri face ca meniul derulant să fie împărțit în două coloane printr-o linie verticală.

Liniile de la 1 la 4 ale listingului 13.3 definesc valorile de identificatori pentru cele patru elemente de meniu care vor fi adăugate. În linia 9 este declarat obiectul `menuPopup` de tip `CMenu`, fiind inițializat ca meniu derulant prin apelul funcției `CreatePopupMenu()` din linia 12.

În linia 15, funcția `AppendMenu()` atașează la meniu un nou element, având eticheta **Red**. Funcția `InsertMenu()` din linia 18 inserează înainte de **Red** un alt element de meniu, eticheta acestuia fiind **Green**. În linia 22 este adăugat un separator prin atașarea unui element de meniu cu ajutorul indicatorului `MF_SEPARATOR`. Elementul **Blue** este adăugat în liniile 25 și 26 împreună cu un marcaj de validare, iar elementul **Yellow** este adăugat în liniile 29 și 30, fiind afișat în dreapta unei linii separatoare verticale.

În fine, în linia 31 este apelată `TrackPopupMenu()` pentru afișarea și controlarea noului meniu derulant.

Puteți să creați funcții de tratare asociate elementelor de meniu dinamice adăugând la sfârșitul hărții de mesaje a clasei reprezentare intrări corespunzătoare identificatorilor pe care i-ați definit:

```
ON_COMMAND(ID_MENU_YELLOW, OnYellow)
```

(Trebuie să aveți grijă ca liniile `#define` pentru identificatori să se afle în cod înaintea hărții de mesaje, astfel încât compilatorul să poată recunoaște acești identificatori.)

În acest caz, funcția de tratare `OnYellow()` va fi apelată atunci când utilizatorul selectează opțiunea de meniu dinamică având identificatorul `ID_MENU_YELLOW`. Implementarea funcției de tratare a comenzii se face în mod normal:

```

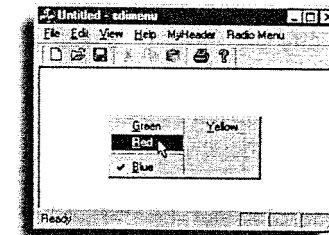
void CSdimenuView::OnYellow()
{
    AfxMessageBox("Yellow");
}

```

Dacă asamblați și rulați aplicația după aplicarea acestor modificări, meniul derulant de context va apărea ca în figura 13.8.

FIGURA 13.8

Un meniu derulant creat dinamic prin intermediul mai multor indicatori de opțiune.



Modificarea dinamică a elementelor de meniu

Puteți, de asemenea, să modificați elemente de meniu în timp ce acestea sunt afișate sau reprezintă obiecte valide, apelând în acest sens funcția `ModifyMenu()`. Această funcție așteaptă patru parametri.

Primul parametru necesar are același rol cu parametrul care specifică poziția pentru `InsertMenu()`. Dacă doriți să specificați poziția prin index, veți transmite ca prim parametru respectivul index pomind de la zero și veți adăuga indicatorul `MF_BYPOSITION` printre indicatorii transmiși sub forma celui de-al doilea parametru. Alternativ, pentru a modifica elementul de meniu care are un anumit identificator, transmiteți acest identificator ca prim parametru și includeți `MF_BYCOMMAND` printre indicatorii care formează al doilea parametru.

Prin cel de-al doilea parametru trebuie transmisă o combinație de indicatori care modifică starea elementului de meniu înlocuind combinația curentă, indicatorii fiind cei prezentați în tabelul 13.2. Așadar, trebuie să transmiteți o nouă combinație de indicatori pentru elementul de meniu pe care vreți să-l modificați.

Puteți (opțional) să transmiteți noul identificator al meniului ca al treilea parametru (sau identificatorul curent pentru a-l lăsa neschimbat), iar prin cel de-al patrulea parametru puteți transmite un pointer la un șir de caractere care va înlocui eticheta curentă a elementului de meniu.

Spre exemplu, pentru a afișa un marcaj de validare alături de elementul de meniu al cărui identificator este `ID_MENU_YELLOW` (creat în secțiunea anterioară), după crearea elementului de meniu ați putea să adăugați următoarea linie de cod:

```
menuPopup.ModifyMenu(ID_MENU_YELLOW,
MF_BYCOMMAND | MF_CHECKED | MF_STRING | MF_MENUBARBREAK,
ID_MENU_YELLOW, "&Yellow");
```

Unele atribute individuale, precum marcajul de validare și starea de activare, pot fi modificate imediat apelând funcțiile `CheckMenuItem()` și `EnableMenuItem()`. Aceste funcții primesc doi parametri. Primul identifică elementul de meniu care va fi modificat, prin identificator sau prin poziția sa pornind de la zero.

Al doilea parametru include unul dintre indicatorii `MF_BYCOMMAND` sau `MF_BYPOSITION` pentru a preciza dacă primul parametru este un identificator sau o valoare index. În cazul funcției `CheckMenuItem()` puteți adăuga `MF_CHECKED` sau `MF_UNCHECKED` pentru a afișa sau, respectiv, înlătura marcajul de validare. Funcția `EnableMenuItem()` permite specificarea unuia dintre indicatorii `MF_ENABLED` sau `MF_DISABLED` pentru a activa sau dezactiva un element de meniu.

De pildă, ați putea să simplificați exemplul cu `ModifyMenu()`, afișând marcajul de validare prin următorul apel al funcției `CheckMenuItem()`:

```
menuPopup.CheckMenuItem(ID_MENU_YELLOW,
MF_BYCOMMAND | MF_CHECKED);
```

Înlăturarea dinamică a elementelor de meniu

Elementele de meniu dintr-un obiect `CMenu` pot fi înlăturate prin intermediul funcției `RemoveMenu()`, aceasta primind doi parametri. Primul parametru este identificatorul sau valoarea de index prin care se precizează elementul de meniu care va fi înlăturat. Al doilea parametru trebuie să fie unul dintre indicatorii `MF_BYCOMMAND` sau `MF_BYPOSITION`, specificând modul în care primul parametru indică poziția.

CAPITOLUL

14

Utilizarea barelor cu instrumente și a barelor de stare

Crearea, personalizarea și utilizarea barelor cu instrumente

Utilizarea barelor de dialog pentru afișarea de controale standard de dialog, cum sunt casetele combinate

Personalizarea barei de stare și afișarea unor informații specifice aplicației

Despre noile bare suport în stilul Internet Explorer

Personalizarea barei cu instrumente standard

Majoritatea aplicațiilor folosesc bare cu instrumente pentru a ajuta utilizatorul să selecteze comenzile frecvent utilizate ale aplicației. De regulă, butoanele de pe bara cu instrumente reprezintă scurtături pentru opțiunile de meniu corespunzătoare, aspectul lor vizual fiind asociat cu comanda în cauză. De exemplu, dacă utilizatorul dorește să tipărească un document, imaginea unei imprimante este o opțiune evidentă.

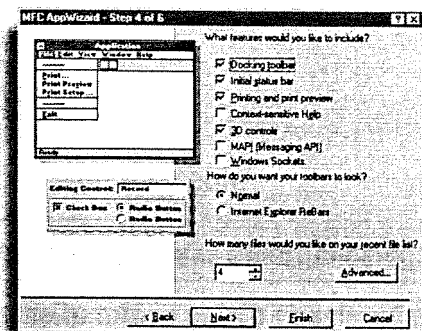
Modificarea barei cu instrumente standard

Bara cu instrumente standard este oferită de AppWizard ca bază de dezvoltare. Puteți să adăugați, să înlăturați sau să modificați butoanele acestora în funcție de nevoile aplicației, sau puteți chiar să înlăturați complet bara cu instrumente dacă nu se dovedește necesară. *În totuși, majoritatea aplicațiilor care lucrează cu fișiere folosesc această bară cu instrumente standard pentru a oferi utilizatorului o interfață comună în cadrul diverselor aplicații Windows.

Atunci când creați un proiect de tip SDI sau MDI, dacă acceptați opțiunile implicite din pasul 4 al dialogului AppWizard, înfățișat în figura 14.1, noua aplicație va fi dotată cu o bară cu instrumente ancorabilă.

FIGURA 14.1

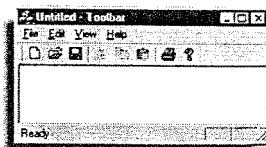
Pasul 4 din MFC AppWizard conține o opțiune pentru crearea unei bare cu instrumente ancorabilă.



AppWizard generează cod care va crea fereastra barei cu instrumente, va încărca imaginile butoanelor dintr-o resursă de tip bară cu instrumente și va atașa bara cu instrumente la fereastra cadru a aplicației. Bara cu instrumente generată de AppWizard conține butoane care corespund opțiunilor implicite din meniurile **File**, **Edit** și **Help**, așa cum se vede în figura 14.2.

FIGURA 14.2

Bara cu instrumente standard dintr-o aplicație SDI generată de AppWizard.



VEDEȚI...

- Pentru mai multe informații despre crearea aplicațiilor SDI, vedeți pagina 246.
- Pentru crearea unei aplicații MDI, vedeți pagina 477.

Despre bara cu instrumente standard

În mod implicit, bara cu instrumente este implementată prin adăugarea unui obiect de tip `CToolBar`, `m_wndToolBar`, în definiția clasei `CMainFrame`, printr-o linie ca aceasta:

```
CToolBar m_wndToolBar;
```

Clasa `CToolbar` este derivată din `CControlBar`, derivată la rândul său din `CWnd`. `CControlBar` dezvoltă funcționalitatea elementară a unei ferestre `CWnd`, adăugând suport pentru ancorarea la o fereastră cadru sau menținerea ferestrei în stare liberă, asemeni unei ferestre obișnuite. Barele cu instrumente, barele de stare și barele de dialog extind apoi `CControlBar` în diferite direcții, particularizând acest suport de fereastră ancorabilă la fiecare dintre aceste tipuri de bare. Clasa `CToolbar` aduce suport pentru un bitmap și un vector de butoane cu scopul de a oferi utilizatorului o interfață de tip bară cu instrumente, asemeni barei standard pe care o înfățișează figura 14.2.

VEDEȚI...

- Pentru amănunte privind clasa `CMainFrame`, vedeți pagina 250.

Crearea barei cu instrumente standard

Vă amintiți că clasa `CMainFrame` este cea care răspunde de gestionarea ferestrei cadru a unei aplicații de tip SDI sau MDI. La crearea ferestrei cadru este apelată funcția membru `OnCreate()` din `CMainFrame`. Codul generat de AppWizard folosește această funcție ca să creeze fereastra barei cu instrumente ca fereastră copil a ferestrei cadru și să încarce resursa de tip bară cu instrumente asociată (`IDR_MAINFRAME`), codul propriu-zis fiind similar celui de aici:

```
if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
    WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)
    || m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
{
    TRACE0("Failed to create toolbar\n");
    return -1;    // fail to create
}
```

Codul de mai sus poate fi găsit în modulul de implementare `MainFrm.cpp`. Crearea ferestrei barei cu instrumente și încărcarea resursei sunt combinate în cadrul condiției `if`, astfel încât dacă una dintre aceste operații eșuează, funcția `OnCreate()` a ferestrei cadru întoarce `-1`.

Funcția `CreateEx()` este folosită pentru a crea fereastra asociată barei cu instrumente. Această funcție membru este nouă în implementarea barelor cu instrumente din Visual C++ 6.0, fiind utilizată pentru a stabili cadrul noilor bare cu aspect plat. Primul parametru transmis este un pointer la fereastra părinte; prin specificarea cuvântului cheie `this` din C++, această fereastră părinte va fi obiectul ferestrei cadru apelant.

Utilizarea funcției CreateEx() pentru crearea barelor cu instrumente în stil Microsoft Internet Explorer 4

Funcția CreateEx() vă permite să stabiliți noi indicatori pentru barele cu instrumente. Pentru crearea unei bare cu instrumente în stilul propus de Microsoft Internet Explorer 4, stabiliți parametrul dwCtrlStyle la TBSTYLE_FLAT | TBSTYLE_TRANSPARENT.

Implementarea generată de AppWizard transmite indicatorul TBSTYLE_FLAT ca al doilea parametru, specificând astfel stilul barei cu instrumente. Dacă preferați stilul clasic, cu butoane „în relief”, puteți să transmiteți aici valoarea zero.

Opțiunile de stil transmise prin intermediul celui de-al treilea parametru al funcției CreateEx() pot fi modificate prin combinarea oricăroră dintre indicatorii prezentați în tabelul 14.1. Trebuie, totuși, să specificați indicatorul de fereastră copil, WS_CHILD, și indicatorul de fereastră vizibilă, WS_VISIBLE.

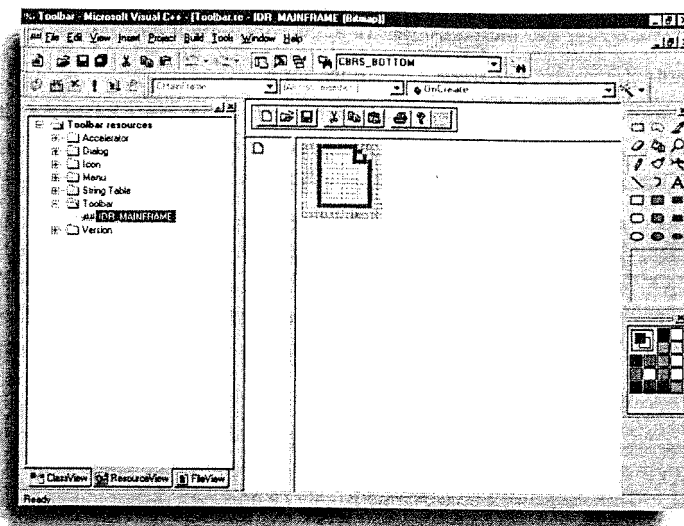
TABELUL 14.1 Indicatori de stil pentru barele de controale

Indicator	Descriere
CBRS_TOP	Ancorează bara cu instrumente în partea superioară a ferestrei cadru, trasând o bordură sub aceasta
CBRS_BOTTOM	Ancorează bara cu instrumente în partea inferioară a ferestrei cadru, trasând o bordură deasupra acesteia
CBRS_LEFT	Ancorează bara cu instrumente în partea stângă a ferestrei cadru, trasând o bordură în dreapta acesteia
CBRS_RIGHT	Ancorează bara cu instrumente în partea dreaptă a ferestrei cadru, trasând o bordură în stânga acesteia
CBRS_ALIGN_ANY	Ancorează bara cu instrumente oriunde
CBRS_FLOAT_MULT	Mai multe bare de controale sunt mobile într-o mini-fereastră cadru
CBRS_TOOLTIPS	Afișează etichete flotante pentru bara de controale
CBRS_FLYBY	Actualizează conținutul mesajului la actualizarea etichetelor flotante
CBRS_GRIPPER	Adaugă o bară de agățare pentru deplasarea barei cu instrumente
CBRS_SIZE_DYNAMIC	Permite utilizatorului să dimensioneze bara cu instrumente atunci când este flotantă
CBRS_SIZE_FIXED	Nu permite utilizatorului să dimensioneze bara cu instrumente atunci când este flotantă

Odată creată fereastra barei cu instrumente, următoarea parte a condiției încarcă resursa de tip bară cu instrumente prin apelul funcției LoadToolBar(), transmițând identificatorul resursei de încărcat. În cazul barei cu instrumente implicite, acest identificator este IDR_MAINFRAME. Resursa de tip bară cu instrumente conține imaginile bitmap pentru butoanele barei cu instrumente și se găsește în pagina ResourceView, în catalogul Toolbar (vedeți figura 14.3).

FIGURA 14.3

Bara cu instrumente SDI/MDI implicită deschisă în cadrul editorului de resurse.



În cazul în care funcția LoadToolBar() reușește să încarce resursa, este întoarsă valoarea TRUE și funcția OnCreate() își continuă execuția în mod normal.

VEDEȚI...

► Pentru mai multe informații despre editorul de resurse, vedeți pagina 45.

Ancorarea barei cu instrumente standard

După crearea barei de stare implicite, funcția OnCreate() se ocupă de ancorarea barei cu instrumente standard. Barele cu instrumente pot fi lăsate libere, asemeni ferestrelor obișnuite, sau pot fi ancorate de o latură a unei ferestre cadru. Ancorarea presupune legarea omogenă a barei cu instrumente la fereastra cadru, fără a fi afișată o fereastră cadru distinctă pentru bara cu instrumente. Prima linie de cod din funcția OnCreate() a ferestrei cadru care se ocupă de ancorare este următoarea:

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
```

Această linie permite barei cu instrumente să fie ancorată la oricare din laturile ferestrei cadru. Dacă doriți ca bara cu instrumente să rămână flotantă (asemeni ferestrei care conține paleta Controls), comentați linia de mai sus, prefixând-o cu simbolul de comentariu //. Alternativ, puteți să determinați ca bara cu instrumente să poată fi ancorată numai la anumite laturi ale ferestrei cadru, prin combinarea indicatorilor prezentați în tabelul 14.2. Aceste valori se combină cu ajutorul operatorului SAU logic, |. De exemplu, puteți modifica apelul EnableDocking() pentru a permite ancorarea barei cu instrumente exclusiv la laturile stângă și dreaptă a ferestrei:

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_LEFT |
    CBRS_ALIGN_RIGHT);
```


TABELUL 14.2 Indicatori pentru ancorarea barelor de controale

Indicator pentru ancorare	Descriere
CBRS_ALIGN_ANY	Permite ancorarea de orice latură a ferestrei cadru
CBRS_ALIGN_TOP	Permite ancorarea de latura superioară a ferestrei cadru
CBRS_ALIGN_BOTTOM	Permite ancorarea de latura inferioară a ferestrei cadru
CBRS_ALIGN_LEFT	Permite ancorarea de latura stângă a ferestrei cadru
CBRS_ALIGN_RIGHT	Permite ancorarea de latura dreaptă a ferestrei cadru

Remarcați că următoarea linie din implementarea standard a funcției `OnCreate()` a ferestrei cadru apelează funcția membru `EnableDocking()` a acestei ferestre, ca aici:

```
EnableDocking(CBRS_ALIGN_ANY);
```

Această linie permite tuturor laturilor ferestrei cadru să joace rolul de *bază de ancorare* pentru barele de controale. Puteți, de asemenea, să transmiteți o combinație a indicatorilor din tabelul 14.2 pentru a stabili sau restrânge bazele de ancorare disponibile în fereastra cadru.

În ultima linie din implementarea standard a funcției `OnCreate()` este apelată funcția membru `DockControlBar()` a ferestrei cadru, cu scopul de a ancora inițial bara cu instrumente la fereastra cadru, ca mai jos:

```
DockControlBar(&m_wndToolBar);
```

În apelul `DockControlBar()` puteți transmite trei parametri. Primul parametru este obligatoriu și reprezintă un pointer la bara de controale pe care doriți să o ancorați. Prin al doilea parametru puteți transmite unul dintre indicatorii enumerați în tabelul 14.3 pentru a specifica poziția de ancorare a barei de controale, sau zero (valoarea implicită) pentru a arăta că bara de controale poate fi ancorată de oricare dintre laturile ferestrei cadru. Cel de-al treilea parametru vă permite să specificați o poziție exactă de ancorare prin intermediul unui pointer la o structură `RECT` care conține coordonatele din ecran ale barei ancorate.

TABELUL 14.3 Indicatori de specificare a poziției de ancorare la ancorarea unei bare cu instrumente prin `DockControlBar()`

Indicator de ancorare	Descriere
AFX_IDW_DOCKBAR_TOP	Ancorează bara de controale la latura superioară a ferestrei cadru
AFX_IDW_DOCKBAR_BOTTOM	Ancorează bara de controale la latura inferioară a ferestrei cadru
AFX_IDW_DOCKBAR_LEFT	Ancorează bara de controale la latura stângă a ferestrei cadru
AFX_IDW_DOCKBAR_RIGHT	Ancorează bara de controale la latura dreaptă a ferestrei cadru

În loc să ancorați bara de controale, ați putea să o lăsați flotantă înlocuind apelul `DockControlBar()` cu un apel al funcției membru `FloatControlBar()` a ferestrei cadru.

`FloatControlBar()` primește de asemenea trei parametri (doi obligatorii și unul opțional). Primul este un pointer la obiectul bară de controale. Prin al doilea parametru puteți transmite un obiect `CPoint` ai cărui membri specifică coordonatele de pe ecran în care va fi plasat colțul stânga sus al barei de controale. Cel de-al treilea parametru, opțional, precizează orientarea barei de controale. Aici puteți transmite `CBRS_ALIGN_TOP` (valoarea implicită) pentru o bară de controale orizontală sau `CBRS_ALIGN_LEFT` pentru o bară de controale verticală.

De pildă, puteți să lansați aplicația având bara cu instrumente flotantă și poziționată în punctul (50,50), înlocuind în acest sens apelul `DockControlBar()` din funcția `OnCreate()` a ferestrei cadru cu linia următoare:

```
FloatControlBar(&m_wndToolBar,CPoint(50,50));
```

Clase terțe pentru extinderea barelor cu instrumente

Există producători de software care oferă biblioteci de clase ce extind în bună măsură funcționalitatea barelor cu instrumente MFC, punând la dispoziție facilități avansate de ancorare și personalizare, asemeni celor întâlnite în chiar Visual Studio. Spre exemplu, capacitatea de tragere și plasare a butoanelor de pe bară cu instrumente, disponibilă după selectarea opțiunii **Customize** din meniul **Tools**, nu este oferită de clasele obișnuite din MFC. În schimb, această funcționalitate extinsă este oferită de producători terți de biblioteci de clase.

Apelarea oricăreia dintre funcțiile `DockControlBar()` sau `FloatControlBar()` duce la afișarea barei cu instrumente, după care inițializarea aplicației continuă în mod normal.

Adăugarea de butoane pe bara cu instrumente prin intermediul editorului de resurse

Adăugarea de butoane pe bara cu instrumente standard se poate face prin intermediul editorului de resurse, urmând pașii de mai jos pentru editarea barei cu instrumente standard `IDR_MAINFRAME`.

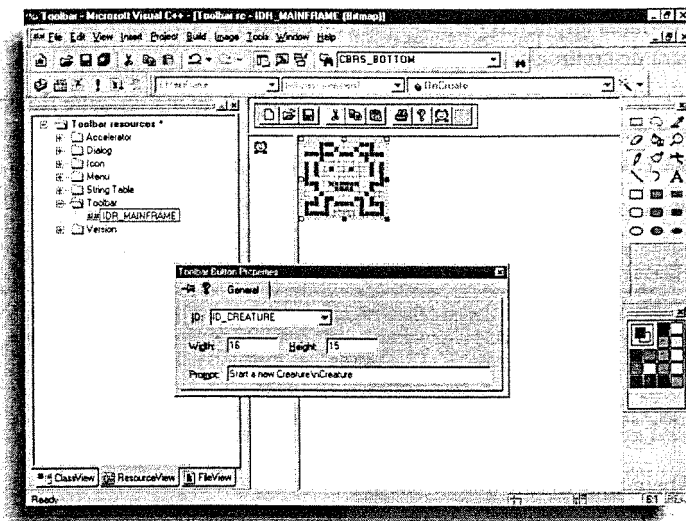
Adăugarea unui buton pe bara cu instrumente

1. Selectați pagina **ResourceView** a secțiunii spațiului de lucru al proiectului.
2. Deschideți arborele de resurse din pagina **ResourceView** pentru a putea vedea resursele de tip bară cu instrumente și efectuați un dublu clic pe bara cu instrumente pe care doriți să o editați.
3. Resursa bară cu instrumente ar trebui să fie afișată într-o fereastră de editare, așa cum o arată figura 14.3.
4. Pentru a adăuga un nou buton, selectați butonul gol afișat în partea dreaptă a barei cu instrumente și începeți să editați bitmap-ul asociat.
5. În momentul în care începeți editarea, veți observa că în dreapta butonului pe care îl editați curent își face apariția un nou buton gol. Dacă doriți să adăugați și alte butoane, puteți să repetați operația de la pasul 4.

6. Pentru fiecare buton nou ar trebui să stabiliți un identificator de comandă și un text explicativ. Acestea pot fi precizate în cadrul casetei de dialog **Toolbar Button Properties** (ilustrată în figura 14.4), care poate fi apelată selectând butonul dorit și apăsând **Alt+Enter** sau efectuând un clic pe meniul **View** și selectând opțiunea **Properties**.

FIGURA 14.4

Stabilirea proprietăților pentru un buton de pe bara cu instrumente.



7. Puteți să stabiliți un identificator de comandă unic, introducând acest nou identificator în caseta combinată **ID**. Ca alternativă, puteți să atribuiți butonului un identificator de comandă asociat unui alt obiect (cum ar fi un identificator de meniu), selectând identificatorul în cauză din lista derulantă a casetei combinate.
8. Puteți specifica un șir de caractere care să fie afișat pe bara de stare, introducând acest șir în caseta de editare **Prompt**. Se poate adăuga, de asemenea, o explicație flotantă, introducând-o în caseta de editare **Prompt** după un simbol **\n** de delimitare.
9. Butonul de pe bara cu instrumente va fi afișat la următoarea asamblare și rulare a programului. Dar acest buton va fi dezactivat și afișat voalat până la adăugarea unei funcții de tratare corespunzătoare. Această funcție se adaugă similar cu o funcție de tratare a unei comenzi de meniu, urmând secvența de pași „Adăugarea unei funcții de tratare pentru o comandă de meniu cu ClassWizard” (aflată în capitolul 13, „Lucrul cu meniuri”).

Odată ce ați inserat butonul pe bara cu instrumente și ați creat funcția de tratare corespunzătoare, puteți să adăugați cod specific aplicației care să fie executat la efectuarea unui clic pe buton. Se poate, totodată, să partajați funcția de tratare cu cea pentru o comandă de meniu, așa cum se și procedează atunci când bara cu instrumente oferă o scurtătură pentru o opțiune de meniu.

Maparea peste butoanele de pe bara cu instrumente a combinațiilor de taste

Puteți să definiți combinații de taste pentru butoanele barei cu instrumente adăugând prin intermediul editorului de resurse intrări corespunzătoare în cadrul tabelii de acceleratori, intrări care să asocieze identificatorul de comandă al unui buton cu combinația de taste dorită. De exemplu, în mod implicit se asociază combinația **Ctrl+N** cu **ID_FILE_NEW** (identificatorul pentru primul buton de pe bara cu instrumente standard).

Spre exemplu, o funcție de tratare simplă pentru bara cu instrumente, care afișează o casetă de mesaj, ar putea să fie inclusă în clasa reprezentare (**CToolBarView**) și să arate astfel:

```
void CToolBarView::OnCreature()
{
    AfxMessageBox("Creature");
}
```

VEDEȚI...

- Detaliile privind editarea imaginilor bitmap se găsesc la pagina 223.
- Pentru mai multe amănunte privind tabela de acceleratori, vedeți pagina 45.

Deplasarea și înlăturarea butoanelor și adăugarea de separatori

Butoanele de pe bara cu instrumente pot fi deplasate prin selectarea butonului vizat și tragerea sa în poziția dorită. Adăugarea sau înlăturarea separatorilor se poate face prin tragerea ușoară a butoanelor spre stânga sau spre dreapta.

Dacă doriți să ștergeți un buton, selectați-l și apoi trageți butonul peste zona de editare a imaginii și plasați-l acolo. Butonul va fi înlăturat de pe bara cu instrumente.

Activarea și dezactivarea butoanelor de pe bara cu instrumente

Butoanele de pe bara cu instrumente pot fi activate și dezactivate prin adăugarea unei funcții de tratare a interfeței cu utilizatorul (UI – user interface). Funcția de tratare UI pentru un buton de pe bara cu instrumente operează la fel ca o funcție de tratare UI pentru un meniu, dar numărul funcțiilor din **CCmdUI** care pot fi apelate, precum **SetText()**, este mai mic.

De ce butoanele de pe bara cu instrumente sunt uneori dezactivate

Funcționalitatea oferită de biblioteca MFC asigură dezactivarea automată a acelor butoane de pe bara cu instrumente care nu au asociată funcție de tratare pentru nici unul din mesajele **COMMAND** sau **UPDATE_COMMAND_UI**.

Pentru a controla starea de activare/dezactivare a unui buton de pe bara cu instrumente, puteți să transmiteți o variabilă membru de control în apelul funcției **Enable()** a obiectului **CCmdUI**, transmis ca pointer funcției de tratare UI corespunzătoare, printr-un cod ca cel de pe pagina următoare:

```
void CToolBarView::OnUpdateCreature(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_bEnableCreature);
}
```

Este posibilă și controlarea stării de apăsare a unui buton, transmițând TRUE sau FALSE în apelul funcției `SetRadio()` a obiectului `CCmdUI`. Valoarea TRUE face ca butonul să apară tot timpul apăsat, în timp ce valoarea FALSE duce la afișarea butonului ridicat, ca de obicei.

VEDEȚI...

➤ Pentru informații privind funcțiile de tratare a interfeței cu utilizatorul, vedeți pagina 277.

Adăugarea unei bare cu instrumente proprii

În cazul aplicațiilor mai mici ar putea fi suficientă simpla ajustare a infrastructurii standard pe care o generează `AppWizard`. În cazul unor aplicații mai mari, însă, ați putea dori să puneți la dispoziție mai multe bare cu instrumente pentru diferite aspecte ale aplicației. Ați putea dori, de asemenea, să permiteți utilizatorului să afișeze sau să înlăture independent barele cu instrumente.

Pentru a crea bitmap-urile asociate butoanelor de pe o bară cu instrumente trebuie să creați o nouă resursă de tip bară cu instrumente. Această nouă resursă va putea fi încărcată apoi într-un nou obiect `CToolbar` încapsulat în fereastra cadru.

Adăugarea unei noi resurse de tip bară cu instrumente

Barele de controale și focusul

O fereastră derivată din `CControlBar`, așa cum este `CToolbar`, nu preia în mod normal focusul de la reprezentarea curentă decât în cazul în care conține un control casetă de editare sau casetă combinată. Într-o astfel de situație ați putea descoperi că focusul nu este cedat înapoi reprezentării atunci când efectuați clic în fereastra acesteia. Soluția este supradefinirea funcției `OnMouseActivate()` în cadrul clasei reprezentare pentru a ceda explicit focusul către fereastra reprezentării.

Inserarea unei noi resurse bară cu instrumente

1. Selectați pagina `ResourceView` pentru a afișa resursele proiectului.
2. Efectuați un clic cu butonul drept pe elementul de la nivelul superior și selectați opțiunea **Insert** a meniului afișat pentru a apela caseta de dialog `Insert Resource`.
3. Selectați **Toolbar** din lista cu tipurile de resurse noi.
4. Efectuați un clic pe butonul **New** pentru a insera noua resursă bară cu instrumente.
5. Noua bară cu instrumente ar trebui să apară în catalogul `Toolbar Resource`, iar fereastra de editare ar trebui să conțină un buton gol pe o bară cu instrumente.
6. Efectuați un clic cu butonul drept pe bara cu instrumente și selectați opțiunea **Properties** a meniului de context pentru a afișa proprietățile acesteia.

7. Acum puteți să modificați identificatorul `IDD_` implicit al resursei cu un nume mai potrivit pentru bara cu instrumente.
8. Dimensionați butonul gol afișat pe bara cu instrumente pentru a ajusta mărimea implicită a barei cu instrumente și a butoanelor acesteia (dacă este necesar).
9. Puteți să începeți editarea butonului inițial și să inserați noi butoane, așa cum o arată secvența de pași „Adăugarea unui buton pe bara cu instrumente”, prezentată mai devreme în acest capitol.

Adăugarea barei cu instrumente în fereastra cadru

Odată adăugată resursa de tip bară cu instrumente, va trebui să creați un obiect `CToolbar` care să încapsuleze noua bară cu instrumente în clasa ferestrei cadru (`CMainFrame`). Declarația noii bare cu instrumente poate fi adăugată în definiția clasei `CMainFrame` (în fișierul `MainFrm.h`), după declarațiile celorlalte bare de controale.

De exemplu, după declararea unui obiect numit `m_wndColorSwipeBar` care corespunde unei noi bare cu instrumente, zona respectivă din definiția clasei `CMainFrame` ar arăta astfel:

```
protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;
    CToolBar m_wndColorSwipeBar; // ** Noua bara cu instrumente
```

În continuare puteți să adăugați codul care creează efectiv bara cu instrumente, plasându-l în funcția `OnCreate()` a ferestrei cadru, după crearea barei cu instrumente standard, printr-un apel `CreateEx()` similar cu cel existent, ca aici:

```
if (!m_wndColorSwipeBar.CreateEx(this, TBSTYLE_FLAT,
    WS_CHILD | WS_VISIBLE | CBRS_LEFT | CBRS_GRIPPER |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
    !m_wndColorSwipeBar.LoadToolBar(IDR_COLORSWIPE))
{
    TRACE0("Failed to create color swipe toolbar\n");
    return -1; // crearea a esuat
}
```

Indicatorii specificați vor duce la crearea unei bare cu instrumente cu aspect plat, ancorată inițial de latura stângă a ferestrei cadru și oferind utilizatorului o bară de agățare care să permită modificarea poziției. Butoanele aflate pe bara cu instrumente și imaginile acestora sunt încărcate din resursa `IDR_COLORSWIPE` de tip bară cu instrumente.

Crearea de bare cu instrumente flotante

Ca alternativă la ancorarea noii bare cu instrumente, puteți oferi o bară cu instrumente flotantă, asemeni paletelor `Controls` pe care ați întâlnit-o în editorul de resurse, apelând în acest scop funcția `FloatControlBar()`.

Dacă noua bară cu instrumente a fost creată cu succes, puteți adăuga liniile de cod care permit și apoi efectuează ancorarea barei cu instrumente:

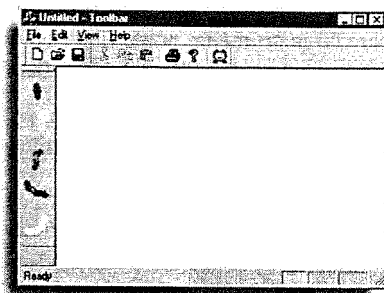
```
m_wndColorSwipeBar.EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndColorSwipeBar);
```

Acum puteți să adăugați funcții de tratare a mesajelor de comandă și a interfeței cu utilizatorul corespunzătoare noii bare cu instrumente.

Spre exemplu, figura 14.5 înfățișează o bară cu instrumente suplimentară, ancorată de latura stângă a ferestrei cadru a unei aplicații SDI. Bara cu instrumente standard a fost modificată și este ancorată de latura superioară.

FIGURA 14.5

O aplicație SDI care afișează o a doua bară cu instrumente, mai mare și ancorată de latura stângă a ferestrei cadru.



VEDEȚI...

► Pentru detalii privind clasa `CMainFrame`, vedeți pagina 250.

Înlăturarea și afișarea barei cu instrumente

Puteți să înlăturați sau să afișați o bară cu instrumente apelând la funcția `ShowControlBar()` a ferestrei cadru. `ShowControlBar()` solicită trei parametri. Primul este un pointer la bara de controale pe care doriți să o înlăturați sau să o faceți vizibilă. Al doilea parametru este o valoare de adevăr care va fi `TRUE` pentru afișarea barei de controale sau `FALSE` pentru înlăturarea acesteia. În ceea ce privește al treilea parametru, transmiteți valoarea `FALSE` pentru a afișa imediat bara de controale sau `TRUE` pentru a întârzia afișarea acesteia.

Localizarea unei bare cu instrumente în cadrul aplicației

În mod normal, barele cu instrumente sunt încapsulate ca membri publici ai clasei `CMainFrame`. Pentru a localiza acest obiect puteți apela funcția globală `AfxGetMainWnd()`. Rețineți că pointerul obținut trebuie convertit la clasa ferestrei cadru a aplicației, de regulă (`CMainFrame*`).

De exemplu, ați putea dori să implementați o opțiune de meniu care să comute vizibilitatea unei bare cu instrumente, afișând un marcaj de validare alături de opțiune atunci când bara este vizibilă. Funcția de tratare a mesajului de comandă pentru această opțiune de meniu s-ar putea prezenta astfel:

```
void CMainFrame::OnViewColorswipeBar()
{
    m_bColorVisible = !m_bColorVisible;
    ShowControlBar(&m_wndColorSwipeBar, m_bColorVisible, FALSE);
}
```

Valoarea variabilei `m_bColorVisible` inserată în obiectul `CMainFrame` este comutată între `TRUE` și `FALSE` cu ajutorul combinației de operatori `!=`. Bara cu instrumente este apoi afișată sau înlăturată transmițând variabila `m_bColorVisible` în apelul funcției `ShowControlBar()`.

Ați putea, de asemenea, să implementați următoarea funcție de tratare UI, care să comute afișarea marcajului de validare alături de opțiunea de meniu în funcție de starea de vizibilitate a barei cu instrumente:

```
void CMainFrame::OnUpdateViewColorswipebar(CCmdUI* pCmdUI)
{
    pCmdUI->Enable();
    pCmdUI->SetCheck(m_bColorVisible);
}
```

VEDEȚI...

► Pentru a afla mai multe despre funcțiile de tratare a interfeței cu utilizatorul corespunzătoare meniurilor, vedeți pagina 276.

Stocarea și încărcarea pozițiilor barelor cu instrumente

Pozițiile tuturor barelor cu instrumente ale unei aplicații pot fi salvate sau încărcate apelând funcțiile `SaveBarState()` și `LoadBarState()` ale ferestrei cadru. Ambele funcții solicită un singur parametru, care este un șir de caractere reprezentând numele unei chei pentru o intrare în registrul sistemului. Transmiteți un șir de caractere unic și pozițiile barelor de controale vor fi salvate (de regulă, la închiderea aplicației). Dacă doriți apoi ca la pornirea aplicației să încărcați pozițiile salvate anterior, va trebui să transmiteți același șir de caractere. Aceste date vor fi stocate în cadrul ramurii implicite din registru care corespunde aplicației (în `HKEY_CURRENT_USER\Software\numelefirmei\numeleaplicației`).

Combinațiile de taste funcționează și în cazul butoanelor dezactivate

Apăsarea unei combinații de taste va genera mesajul de comandă corespunzător chiar dacă butonul asociat de pe bara cu instrumente este dezactivat. Așa stând lucrurile, va trebui să țineți cont de posibilitatea ca funcția de tratare a comenzii să fie apelată chiar dacă butonul de pe bara cu instrumente este dezactivat.

De exemplu, ați putea să salvați starea curentă a barelor de controale înainte de distrugerea ferestrei cadru, adăugând o funcție de tratare pentru mesajul `WM_DESTROY` și efectuând salvarea înainte de a apela funcția `OnDestroy()` din clasa de bază:

```
void CMainFrame::OnDestroy()
{
    ...
}
```

```
SaveBarState("MyToolBarSettings");
CFrameWnd::OnDestroy();
}
```

(Pentru adăugarea acestei funcții de tratare, consultați secvența de pași „Adăugarea unei funcții de tratare pentru mesajul WM_DESTROY” din capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”).

Pentru a încărca apoi datele salvate, după crearea barelor cu instrumente, puteți să adăugați un apel corespunzător la sfârșitul funcției `OnCreate()` a ferestrei cadru:

```
LoadBarState("MyToolBarSettings");
```

Atunci când aplicația este lansată din nou, pozițiile și stările de ancorare ale barelor cu instrumente care au fost stabilite mai devreme de utilizator vor fi puse din nou în vigoare.

Utilizarea barelor de dialog

Barele de dialog sunt un tip de bare cu instrumente care au la bază casete de dialog nemodale. Puteți să adăugați orice controale într-o machetă de dialog și să folosiți apoi acea machetă pentru a afișa o bară de dialog. În mod normal este folosită macheta implicită `IDD_DIALOGBAR`, care este o machetă de dialog cu aspect de bară, fără butoanele modale `OK` și `Cancel`.

O bară de dialog poate fi întâlnită în implementarea standard a previzualizării. Aceasta conține șirul de butoane **Print**, **Next Page**, **Prey Page** etc.

Încărcarea și crearea unei bare de dialog se efectuează ca în cazul unei bare cu instrumente, prin încapsularea unui obiect `CDialogBar` într-o fereastră cadru. `CDialogBar` este derivată din `CControlBar`, deci moștenește toată funcționalitatea legată de ancorare, permițând barelor de dialog să fie ancorate și să se comporte asemeni barelor cu instrumente.

Utilizarea butoanelor grafice în barele de dialog

Adăugarea într-o bară de dialog a unor butoane „normale” de bară cu instrumente (cu imagini) este mai dificilă. Dacă doriți să adăugați butoane cu imagini, este de preferat să folosiți butoane grafice, apelând la clasa `CBitmapButton` așa cum am arătat în secțiunea „Crearea butoanelor grafice” din capitolul 11.

VEDEȚI...

- Pentru a afla despre casetele de dialog nemodale, vedeți pagina 213.
- Pentru mai multe amănunte privind tipărirea și previzualizarea, vedeți pagina 504.

Adăugarea unei resurse de tip bară de dialog

Adăugarea unei resurse machetă de dialog pentru o bară de dialog se poate face urmând secvența de pași „Inserarea unei noi resurse de tip machetă de dialog” (aflată în capitolul 10, „Utilizarea casetelor de dialog”), selectând în caseta de dialog **Insert Resource** macheta de dialog `IDD_DIALOGBAR`.

Înlăturarea marcajelor de ghidare la editarea unei bare de dialog

În cazul în care folosiți macheta implicită `IDD_DIALOGBAR`, ați putea dori să înlăturați marcajele de ghidare de pe suprafața casetei de dialog pentru a utiliza la maxim suprafața oricum limitată a unei bare cu instrumente. În acest scop trebuie să efectuați un clic pe meniul **Layout** și să selectați opțiunea **Guide Settings** pentru a afișa caseta de dialog **Guide Settings**. Aici puteți selecta opțiunea **None** din interiorul cadrului **Layout Guides** pentru a înlătura orice marcaje de ghidare.

După inserarea noii resurse machetă de dialog și modificarea corespunzătoare a identificatorului și a proprietăților acesteia, vă puteți ocupa de adăugarea de controale, exact ca în cazul unei casete de dialog obișnuite.

Este bine să stabiliți identificatori pentru fiecare control adăugat, astfel încât să puteți accesa aceste controale *prin cod*.

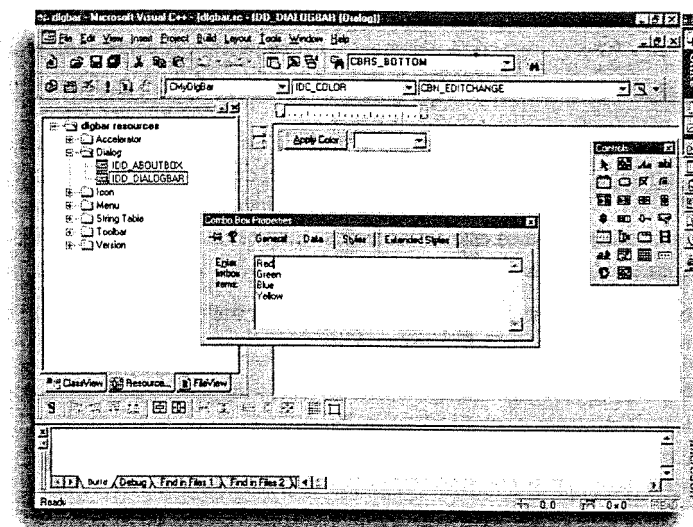
Figura 14.6 înfățișează o astfel de machetă de dialog pentru o bară de dialog. Macheta conține două controale, un buton (etichetat **Apply Color**) și o casetă combinată cu listă derulantă care este inițializată cu o listă de culori.

VEDEȚI...

- Pentru a afla mai multe despre machetele pentru casetele de dialog, vedeți pagina 54.

FIGURA 14.6

Editarea unei resurse machetă pentru o bară de dialog.



Adăugarea barei de dialog în fereastra cadru

Adăugarea unei bare de dialog într-o fereastră cadru se face ca în cazul unei bare cu instrumente, declarând un obiect `CDialogBar` în clasa `CMainFrame`. De exemplu, pentru a declara un obiect public de tip `CDialogBar`, numit `m_wndColorDlgBar`, ați putea adăuga în definiția clasei `CMainFrame` linia următoare:

```
public:
    CDialogBar m_wndColorDlgBar;
```

Această linie poate fi adăugată direct prin editare sau prin intermediul casetei de dialog Add Member Variable.

Noua bară de dialog poate fi creată prin apelarea funcției `Create()` a obiectului `CDialogBar`, în interiorul funcției `OnCreate()` a ferestrei cadru. Funcția `Create()` a unei bare de dialog solicită patru parametri. Primul dintre aceștia specifică fereastra părinte a barei de dialog (de regulă, fereastra cadru, transmisă din funcția `OnCreate()` prin intermediul pointerului `this` din C++). Macheta barei de dialog se transmite prin cel de-al doilea parametru, indicând astfel macheta care va fi încărcată și afișată de către bara de dialog. Starea de ancorare inițială poate fi stabilită transmitând unul dintre indicatorii de aliniere prezentați în tabelul 14.4.

TABELUL 14.4 Indicatori de aliniere pentru bara de dialog

Indicator	Descriere
CBRS_TOP	Ancorează bara de dialog în partea superioară a ferestrei cadru
CBRS_BOTTOM	Ancorează bara de dialog în partea inferioară a ferestrei cadru
CBRS_LEFT	Ancorează bara de dialog în partea stângă a ferestrei cadru
CBRS_RIGHT	Ancorează bara de dialog în partea dreaptă a ferestrei cadru
CBRS_NOALIGN	Bara de dialog nu este afectată de dimensionarea ferestrei cadru

Al patrulea parametru vă permite să atribuiți un identificator pentru bara de dialog propriu-zisă. Aici puteți specifica valori aflate în intervalul definit de `AFX_IDW_CONTROLBAR_FIRST` și `AFX_IDW_CONTROLBAR_LAST`. Primii câțiva identificatori sunt utilizați pentru implementarea previzualizării, așa că este de preferat să alegeți valori obținute prin scăderea unor valori incrementale din `AFX_IDW_CONTROLBAR_LAST`, evitând astfel eventualele conflicte.

De exemplu, pentru a crea fereastra barei de dialog pentru obiectul `m_wndColorDlgBar`, ați putea să adăugați în funcția `OnCreate()` a ferestrei cadru următoarele linii de cod:

```
if (!m_wndColorDlgBar.Create(this, IDD_DIALOGBAR,
    CBRS_TOP, AFX_IDW_CONTROLBAR_LAST-1))
{
    TRACE0("Failed to create dialog bar\n");
    return -1;    // crearea a esuat
}
```

Dacă crearea reușește, puteți stabili ancorarea barei de dialog, exact așa cum ați proceda în cazul oricărei alte bare de controale, prin intermediul liniilor următoare:

```
m_wndColorDlgBar.EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndColorDlgBar);
```

VEDEȚI...

➤ Pentru detalii privind clasa `CMainFrame`, vedeți pagina 250.

Tratarea controalelor de pe bara de dialog

Mesajele generate de controalele aflate pe bara de dialog se tratează prin adăugarea unor intrări corespunzătoare în hărțile de mesaje și a unor funcții de tratare, fie în clasa ferestrei cadru, fie în clasele reprezentărilor.

Pentru adăugarea acestor intrări puteți utiliza `ClassWizard`, selectând controlul dorit de pe bara de dialog în cadrul editorului de resurse, apelând `ClassWizard` prin combinația de taste `Ctrl+W` și urmând pașii cunoscuți pentru adăugarea unei funcții de tratare pentru un control. Diferența față de cazul unei casete de dialog obișnuite constă în faptul că funcțiile de tratare trebuie implementate într-o clasă reprezentare sau document, și nu într-o clasă dialog (asemeni unei funcții de tratare pentru un meniu sau o bară cu instrumente).

Spre exemplu, listingul 14.1 ilustrează intrările din harta de mesaje și funcțiile de tratare pentru controalele aflate pe bara de dialog din figura 14.6. Identificatorul controlului buton este `IDC_APPLY_COLOR`, iar identificatorul casetei combinate cu listă derulantă este `IDC_COLOR`. Codul permite utilizatorului să modifice culoarea de fond a reprezentării prin selectarea unei culori din lista derulantă și efectuarea unui clic pe butonul **Apply** pentru redesenarea ferestrei, așa cum se vede în figura 14.7.

LISTINGUL 14.1 LST15_1.CPP – Implementarea într-o clasă reprezentare a funcțiilor de tratare pentru controalele aflate pe o bară de dialog

```
1 BEGIN_MESSAGE_MAP(CDlgbarView, CView)
2     //{AFX_MSG_MAP(CDlgbarView)
3     ON_WM_ERASEBKGD()
4     //}}AFX_MSG_MAP
5     // Standard printing commands
6     ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
7     ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
8     ON_COMMAND(ID_FILE_PRINT_PREVIEW,
9         CView::OnFilePrintPreview)
9     ON_COMMAND(IDC_APPLY_COLOR, ApplyColor)
10    ON_CBN_SELCHANGE(IDC_COLOR, ColorChange) ①
11 END_MESSAGE_MAP()
12
13 afx_msg void CDlgbarView::ApplyColor()
14 {
15     Invalidate();
16 }
17
18 #include "MainFrm.h"
19 afx_msg void CDlgbarView::ColorChange()
20 {
21     CMainFrame* pMainFrame =
22
```

(continuare)

LISTINGUL 14.1 Continuare

```

22     (CMainFrame*)GetParentFrame();
23     CComboBox* pColSel = (CComboBox*) — ❷
24     (pMainFrame->
25         m_wndColorDlgBar.GetDlgItem(IDC_COLOR));
26     if (pColSel) m_nColor = pColSel->GetCurSel();
27 }
28
29 BOOL CDlgbarView::OnEraseBkgnd(CDC* pDC)
30 {
31     COLORREF rf = RGB(255,255,255);
32     switch(m_nColor)
33     {
34     case 0: rf = RGB(0,0,255); break;
35     case 1: rf = RGB(0,255,0); break;
36     case 2: rf = RGB(255,0,0); break;
37     case 3: rf = RGB(255,255,0); break;
38     }
39     CRect rcClient;
40     GetClientRect(&rcClient);
41     pDC->FillSolidRect(&rcClient, rf);
42     return TRUE;
43 }

```

- ❶ Intrare adăugată manual în harta de mesaje pentru a răspunde la mesajul de modificare a selecției în caseta combinată de pe bara de dialog.
- ❷ Controlul casetă combinată este preluat de pe bara de dialog și mapat dinamic pe o clasă control în scopul determinării selecției curente.
- ❸ În funcție de elementul selectat curent, se alege o valoare COLORREF pentru colorarea fundalului reprezentării la primirea unui mesaj WM_ERASEBKND.

Linia 9 din listingul 14.1 reprezintă intrarea din harta de mesaje care corespunde controlului buton (al cărui identificator este IDC_APPLY_COLOR). Această intrare din harta de mesaje asociază mesajul de comandă generat de buton cu funcția de tratare ApplyColor() (exact așa cum s-ar întâmpla în cazul unui buton obișnuit de pe o bară cu instrumente). Intrarea în harta de mesaje din linia 10 ilustrează asocierea unei funcții de tratare (ColorChange()) cu mesajul CBN_SELCHANGE, trimis la modificarea selecției din cadrul casetei combinate. După adăugarea acestei intrări din harta de mesaje (linia 10), modificarea selecției în caseta combinată de pe bara de dialog va avea ca efect apelarea funcției de tratare ColorChange().

Funcția de tratare ApplyColor(), implementată între liniile 13 și 16, forțează redesenarea ferestrei prin apelul Invalidate() din linia 15.

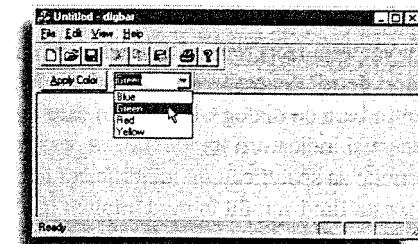
Funcția de tratare ColorChange(), implementată în liniile 19-27, modifică variabila membru m_nColor, care reține indicele culorii selectate, în funcție de opțiunea selectată de utilizator din listă. După obținerea unui pointer la fereastra cadru, în linia 21, caseta combinată este mapată dinamic pe baza pointerului său de fereastră asociat, care se determină apelând funcția GetDlgItem() a barei de dialog și transmitând identificatorul IDC_COLOR pentru a preciza controlul casetă combinată. În cazul în care pointerul obținut este valid, culoarea selectată curent este determinată prin apelul funcției GetCurSel() a controlului.

Observați că în linia 18 este inclus modulul MainFrm.h, făcând astfel cunoscut compilatorului obiectul m_wndColorDlgBar declarat în definiția CMainFrame.

Liniile 29-43 constituie funcția de tratare a mesajului WM_ERASEBKND. Culoarea corespunzătoare este stabilită în liniile 31-38, iar fundalul este colorat prin apelul funcției FillSolidRect() din linia 41.

FIGURA 14.7

O aplicație SDI care afișează o bară de dialog conținând o casetă combinată.



VEDEȚI...

- Pentru a înțelege mesajele generate de casetele combinate, vedeți pagina 124.
- Pentru mai multe amănunte privind culorile și macrodefiniția RGB, vedeți pagina 350.
- Pentru mai multe informații despre trasarea dreptunghiurilor cu ajutorul pensulelor, vedeți pagina 372.

Personalizarea barei de stare

În bara de stare a unei aplicații SDI sau MDI puteți să adăugați *indicatori* proprii. Implementarea standard oferă în partea inferioară a ferestrei cadru indicatori pentru starea tastelor CAPS Lock, NUM Lock și SCRL Lock. Aceștia sunt afișați sub forma unor casete adâncite, numite *casete de indicare*.

Convenții privind bara de stare

Prin convenție, bara de stare este ancorată în partea inferioară a ferestrei cadru a aplicației și nu poate fi deplasată. Aceasta este funcționalitatea implicită în cazul în care nu interveniți asupra indicatorilor transmiși la creare. Firește, sunteți liber să depășiți convențiile și să utilizați mecanismele pe care le oferă barele de controale așa cum găsiți de cuviință.

Aceste casete pot fi create, accesate și întreținute prin intermediul unui obiect CStatusBar declarat în fereastra cadru principală. CStatusBar este o altă clasă derivată din

CControlBar și, prin urmare, dispune de întreaga funcționalitate de ancorare pe care o oferă barele de controale.

VEDEȚI...

► Pentru mai multe detalii privind aplicațiile SDI și MDI, vedeți pagina 480.

Despre bara de stare standard

Codul de implementare standard pe care AppWizard îl generează ca suport pentru bara de stare este foarte similar cu codul pentru bara cu instrumente. În clasa CMainFrame se găsește o declarație a unui obiect CStatusBar, numit m_wndStatusBar:

```
CStatusBar m_wndStatusBar;
```

Fereastra barei de stare se creează apoi în funcția OnCreate() a ferestrei cadru, prin apelul funcției Create() a clasei CStatusBar. Funcția Create() a barei de stare poate primi trei parametri. Numai primul parametru, un pointer la fereastra cadru părinte, este obligatoriu. Al doilea parametru vă permite să transmiteți indicatori care să specifice poziția de ancorare și, eventual, indicatori de stil pentru fereastră. Aici pot fi transmiși indicatorii de poziție din tabelul 14.4 (pentru bara de dialog). În cazul în care omiteți acest al doilea parametru, în mod implicit sunt transmiși indicatorii WS_CHILD, WS_VISIBLE și CBSR_BOTTOM. Al treilea parametru vă permite să specificați un identificator al barei de stare, similar cu cel pentru bara de dialog, acesta fiind stabilit în mod implicit la AFX_IDW_STATUS_BAR.

În implementarea standard, puteți observa că bara de stare este creată după bara cu instrumente, în funcția OnCreate() a ferestrei cadru, prin următoarea secvență de cod:

```
if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
    sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1;    // fail to create
}
```

Dacă apelul Create reușește, este apelată funcția SetIndicators() a barei de stare în scopul inițializării casetelor de indicare. Această funcție primește un pointer la un vector de valori UINT (întreg fără semn) ca prim parametru și numărul indicatorilor transmiși ca al doilea parametru. Codul de implementare standard, prezentat mai devreme, transmite un vector numit indicators și folosește operatorul sizeof() pentru a determina numărul de elemente, împărțind dimensiunea totală a vectorului indicators la dimensiunea unui element.

Activarea etichetelor explicative pentru bara de stare

Pentru a utiliza etichete explicative în cadrul unei bare de stare, va trebui să fixați indicatorii de stil ai obiectului CStatusBarCtrl aflat la bază. Aceasta se poate face înlocuind apelul Create() cu apelul funcției CreateEx() a clasei CStatusBar. Al doilea parametru pentru CreateEx() este dwCtrlStyle, iar prin intermediul acestuia puteți să transmiteți indicatorul SBT_TOOLTIPS, care activează etichetele explicative.

Vectorul indicators este definit la începutul modulului de implementare MainFrm.cpp, înaintea funcțiilor constructor și OnCreate() ale ferestrei cadru. Codul generat în mod implicit de către AppWizard pentru declararea vectorului indicators este următorul:

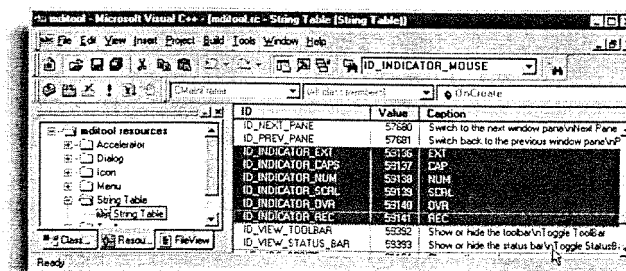
```
static UINT indicators[] =
{
    ID_SEPARATOR,        // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

Primul element, ID_SEPARATOR, specifică un separator care are ca efect generarea în cadrul barei de stare a unui segment normal (care nu este adâncit). Implicit, primul element din vector se întinde pe spațiul lăsat de ceilalți indicatori. Pentru că acest prim element este un separator, aplicația va afișa un segment de fereastră obișnuit care se întinde până la ceilalți indicatori. Acest segment va fi utilizat mai târziu pentru a afișa mesaje explicative privind opțiunile de meniu sau butoanele de pe bara cu instrumente atunci când vă aflați în dreptul acestora.

Al doilea element, ID_INDICATOR_CAPS, este un identificator generat de AppWizard care duce la afișarea stării curente a tastei Caps Lock. ID_INDICATOR_NUM și ID_INDICATOR_SCRL fac același lucru pentru tastele Num Lock și Scroll Lock. Acești identificatori generați de AppWizard pot fi găsiți în resursa tabelă de șiruri a aplicației. Pentru fiecare element există un text asociat (așa cum se vede în figura 14.8) care este afișat în caseta de indicare corespunzătoare, în cazul în care aceasta este activată. Casetele de indicare pot fi activate sau dezactivate prin intermediul unor funcții de tratare UI similare cu cele asociate meniurilor sau butoanelor de pe bara cu instrumente. În urma activării este afișat textul asociat, iar în urma dezactivării acest text este șters. Implementarea implicită activează sau dezactivează acești indicatori generați de AppWizard pentru a afișa textul asociat (CAPS, NUM sau SCRL) atunci când tastele Caps Lock, Num Lock și Scroll Lock își modifică starea.

FIGURA 14.8

Elementele selectate în cadrul tabelii de șiruri reprezintă indicatorii generați de AppWizard pentru bara de stare.



Puteți să adăugați, să înlăturați sau să modificați elementele implicite ale vectorului indicators pentru a înlocui indicatorii implicați de pe bara de stare, așa cum vom vedea în

secțiunea următoare. Atunci când acest vector este transmis funcției `SetIndicators()`, elementele sale sunt folosite pentru a modifica aspectul și funcționalitatea barei de stare. După apelarea funcțiilor `Create()` și `SetIndicators()` va fi afișată bara de stare, ancorată la baza ferestrei cadru datorită valorilor implicite ale indicatorilor în funcția `Create()`.

VEDEȚI...

➤ Pentru explicații legate de tabelele de șiruri, vedeți pagina 45.

Adăugarea indicatorilor și a separatorilor

Pentru adăugarea pe bara de stare a unui indicator propriu va trebui mai întâi să inserați un identificator unic în tabela de șiruri. Textul asociat identificatorului în cadrul tabelii de șiruri va fi afișat de către respectivul indicator de pe bara de stare atunci când acesta este activat. Un indicator poate fi activat transmitând valoarea `TRUE` funcției `Enable()` a obiectului `CCommandUI`, obiect primit de către funcția de tratare UI asociată indicatorului. Puteți, de asemenea, să modificați acest text dacă este necesar, apelând funcția `SetText()` a obiectului `CCommandUI` asociat (așa cum ați proceda în cazul unei opțiuni de meniu).

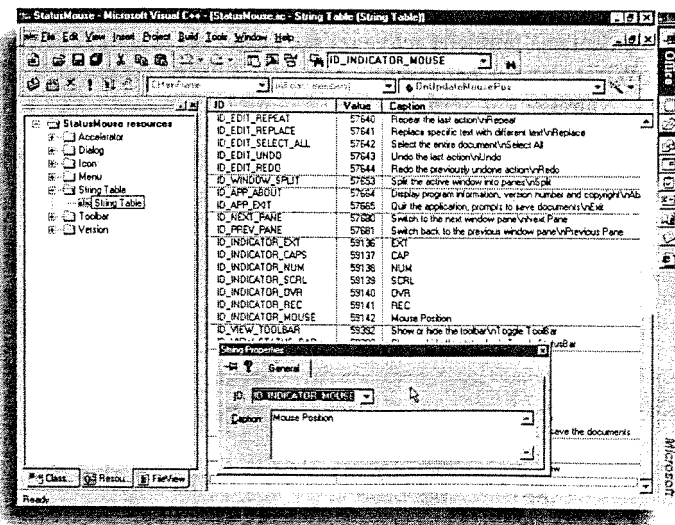
Din păcate, aceste funcții de tratare UI nu pot fi create prin intermediul `ClassWizard`, așa că veți fi nevoit să adăugați manual intrările necesare în harta de mesaje și să vă ocupați de declararea și implementarea funcțiilor propriu-zise.

Un exemplu concret care să afișeze poziția curentă a mouse-ului prin intermediul unui indicator de pe bara de stare ar putea să clarifice întregul proces și elementele necesare. Apelați la `AppWizard` pentru a genera o infrastructură SDI obișnuită (numită `StatusMouse` în codul prezentat).

Pentru început ați putea să creați în tabela de șiruri o intrare corespunzătoare noului indicator, alegând de exemplu identificatorul `ID_INDICATOR_MOUSE` (vedeți figura 14.9).

FIGURA 14.9

Adăugarea în tabela de șiruri a unei noi intrări pentru afișarea poziției mouse-ului.



Adăugarea în tabela de șiruri a unui nou element

1. Selectați pagina `ResourceView` pentru a afișa resursele proiectului.
2. Efectuați un clic pe semnul(-ele) plus pentru a deschide arborele de resurse astfel încât tabela de șiruri vizată să fie afișată în pagina `ResourceView` (vedeți figura 14.9).
3. Efectuați un dublu clic pe `String Table` pentru a afișa elementele tabelii de șiruri în cadrul ferestrei de editare.
4. Derulați în sus sau în jos lista elementelor din tabela de șiruri pentru a găsi secțiunea în care veți adăuga noul element.
5. Efectuați un clic pe ultimul element al secțiunii pentru a-l selecta și apăsați tasta `Insert` pentru a insera un nou element în tabela de șiruri. Ar trebui să fie afișată caseta de dialog `String Properties`.
6. Introduceți în caseta `Caption` textul asociat noului element din tabela de șiruri.
7. Efectuați un clic în caseta `ID` a casetei de dialog `String Properties` și înlocuiți identificatorul `IDS_STRING` cu propriul identificator.
8. Apăsați `Enter` pentru a închide caseta de dialog `String Properties`. Noul element ar trebui să apară acum în lista elementelor din tabela de șiruri.

Odată creat un identificator unic (cum ar fi `ID_INDICATOR_MOUSE`) asociat unei resurse șir de caractere, puteți să-l adăugați în vectorul de indicatori ai barei de stare astfel încât să fie afișat imediat după separatorul care înfățișează texte explicative, ca aici:

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_MOUSE,
    ID_INDICATOR_CAPS,
    ...
}
```

Noul identificator (`ID_INDICATOR_MOUSE`) din tabela de șiruri va identifica de acum o casetă de indicare de pe bara de stare.

Pentru adăugarea unei funcții de tratare UI asociată noului identificator, trebuie mai întâi să creați o nouă intrare în harta de mesaje a ferestrei cadru (modulul `MainFrm.cpp`), realizând astfel asocierea dintre identificator și funcția de tratare:

```
ON_UPDATE_COMMAND_UI(ID_INDICATOR_MOUSE, OnUpdateMousePos)
```

Linia de mai sus stabilește asocierea între identificatorul `ID_INDICATOR_MOUSE` și funcția de tratare `OnUpdateMousePos()`. Intrările adăugate manual în harta de mesaje trebuie plasate după cele generate de `ClassWizard` (adică după comentariul `// AFX_MSG_MAP` de închidere). În continuare, funcția de tratare UI ar putea să arate astfel:

```
void CMainFrame::OnUpdateMousePos(CCommandUI* pCmdUI)
{
    pCmdUI->Enable();
}
```

Funcția `Enable()` stabilește starea implicită de activare (`TRUE`), determinând afișarea textului asociat și având ca rezultat o bară de stare conținând un nou indicator în care apare șirul **Mouse Position** (din tabela de șiruri). Dacă tot ce doriți este să comutați starea de activare a acestui indicator, puteți să transmiteți o variabilă membru de tip `Boolean` în apelul funcției `Enable()` (așa cum ați făcut în secțiunea „Activarea și dezactivarea butoanelor de pe bara cu instrumente”, mai devreme în acest capitol). În loc să ducă la afișarea voalată a indicatorului, așa cum se întâmplă cu butoanele de pe bara cu instrumente și cu opțiunile de meniu, dezactivarea unui indicator de pe bara de stare are ca efect dispariția textului afișat.

Pentru că funcția de tratare UI a fost creată manual, trebuie să adăugați o declarație de funcție corespunzătoare în definiția clasei `CMainFrame` (aflată în fișierul `MainFrm.h`). Declarația pentru funcția de tratare de mai sus ar arăta astfel:

```
afx_msg void OnUpdateMousePos(CCmdUI* pCmdUI);
```

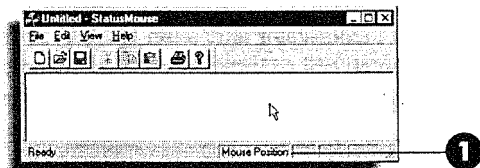
(Prefixul `afx_msg` este utilizat de macrodefinițiile din harta de mesaje.)

Prin urmare, noua casetă de indicare va apărea alături de celelalte casete, în interiorul său fiind afișat șirul **Mouse Position** (vedeți figura 14.10).

FIGURA 14.10

Un nou indicator pe bara de stare, afișând șirul **Mouse Position**.

- 1 Un nou indicator pe bara de stare, afișând șirul **Mouse Position**.



Puteți să adăugați în vectorul de indicatori elemente `ID_SEPARATOR` care să ducă la afișarea de separatori între casetele de indicare. De exemplu, dacă adăugați un `ID_SEPARATOR` după elementul `ID_INDICATOR_MOUSE` al vectorului de indicatori:

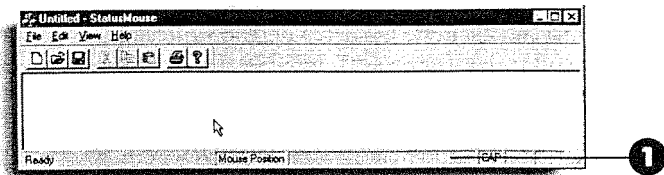
```
ID_SEPARATOR,          // status line indicator
ID_INDICATOR_MOUSE,
ID_SEPARATOR,          // ** noul separator
ID_INDICATOR_CAPS,
```

Va fi afișată o nouă casetă de indicare ce separă indicatorii **Mouse Position** și **CAP** (vedeți figura 14.11).

FIGURA 14.11

Efectul adăugării unui nou separator după indicatorul **Mouse Position**.

- 1 Noua casetă separator.



În orice caz, probabil că veți dori să micșorați dimensiunile inițiale ale noului separator și să-i modificați aspectul, așa cum vom vedea în secțiunea care urmează.

VEDEȚI...

- Pentru a afla mai multe despre funcțiile de tratare a interfeței cu utilizatorul în cazul meniurilor, vedeți pagina 276.
- Pentru explicații privind tabelele de șiruri, vedeți pagina 45.

Modificarea dinamică a dimensiunii, stilului și conținutului casetelor de indicare

Clasa `CStatusBar` conține o serie de funcții ce vă permit să modificați dimensiunile, stilul și conținutul textual al indicatorilor. Aceste funcții sunt `SetPaneInfo()`, `SetPaneStyle()` și, respectiv, `SetPaneText()`. Fiecare funcție identifică o casetă de indicare prin poziția acesteia pornind de la zero în cadrul vectorului de indicatori (transmis în `SetIndicators()`).

Specificarea directă a poziției nu este întotdeauna soluția potrivită, deoarece ați putea fi nevoit să modificați o mulțime de linii de cod dacă inserați un nou indicator în mijlocul barei de stare (pozițiile existente se vor schimba). Pentru a vă veni în ajutor, funcția membru `CommandToIndex()` furnizează poziția unui indicator pe baza identificatorului aceluia indicator, transmis ca unic parametru.

De exemplu, linia următoare înscrie în variabila `nIndex` valoarea 1 pe care o întoarce funcția `CommandToIndex` (considerând vectorul de indicatori `indicators` precizat anterior).

```
int nIndex =
    m_wndStatusBar.CommandToIndex(ID_INDICATOR_MOUSE);
```

Acum puteți insera noi elemente înainte de `ID_INDICATOR_MOUSE`, iar valoarea `nIndex` se va modifica pentru a reflecta poziția corectă a indicatorului dorit, poziție pe care o veți putea transmite apoi funcțiilor `SetPane...` ().

Funcția `SetPaneText()` primește ca prim parametru poziția indicatorului al cărui conținut doriți să-l modificați. Al doilea parametru este un pointer către noul conținut al indicatorului, acesta putând fi un șir terminat cu caracterul nul sau un obiect `CString`. Cel de-al treilea parametru (opțional) este o valoare de adevăr care arată dacă indicatorul va fi invalidat în scopul afișării imediate a noului conținut (valoarea `TRUE` implicită) sau dacă acesta va fi actualizat cu ocazia următoarei redesenări a ferestrei (valoarea `FALSE`).

De pildă, linia de cod următoare schimbă conținutul indicatorului aflat în poziția `nIndex`:

```
m_wndStatusBar.SetPaneText(nIndex, "My New Pane Text");
```

Funcția reciprocă `GetPaneText()` întoarce o variabilă `CString` în care este înscris conținutul curent al indicatorului specificat prin poziția transmisă ca parametru.

Stilul unei casete de indicare poate fi modificat apelând funcția `SetPaneStyle()` și transmitând în al doilea parametru un indicator de stil (dintre cei prezentați în tabelul 14.5), primul parametru identificând indicatorul prin poziție.

Spre exemplu, puteți să determinați afișarea în relief a indicatorului **Mouse Position** prin intermediul următoarelor linii de cod:

```
int nIndex = CommandToIndex(ID_INDICATOR_MOUSE);
m_wndStatusBar.SetPaneStyle(nIndex, SBPS_POPOUT);
```

TABELUL 14.5 Indicatori de stil folosiți pentru stabilirea aspectului casetelor de indicare

Indicator de stil	Descriere
SBPS_NORMAL	O casetă de indicare obișnuită, cu aspect adâncit
SBPS_POPOUT	Caseta de indicare apare ieșită în relief, nu adâncită
SBPS_DISABLED	Conținutul textual al indicatorului nu mai este afișat
SBPS_STRETCH	Caseta de indicare ocupă tot spațiul rămas pe bara de stare (un singur indicator poate avea fixat acest indicator)
SBPS_NOBORDERS	Caseta de indicare nu mai are aspect 3D

Funcția `GetPaneStyle()` întoarce o valoare `UINT` care conține indicatorii de stil pentru caseta de indicare specificată prin poziția sa transmisă ca parametru.

Identificatorul, stilul și lățimea unei casete de indicare pot fi modificate la un loc apelând funcția `SetPaneInfo()`. Ca de obicei, primul parametru reprezintă poziția indicatorului modificat. Prin cel de-al doilea parametru puteți să transmiteți un identificator care să înlocuiască identificatorul curent (sau același identificator pentru a-l menține neschimbat). Al treilea parametru vă permite să transmiteți oricare dintre indicatorii de stil prezentați în tabelul 14.5 cu scopul de a ajusta stilul casetei de indicare. În fine, al patrulea parametru vă permite să modificați lățimea implicită a casetei de indicare prin specificarea unei valori întregi care reprezintă noua lățime a casetei, exprimată în pixeli (această valoare este ignorată în cazul în care este fixat indicatorul de stil `SBPS_STRETCH` al casetei).

De exemplu, liniile de cod de mai jos modifică lățimea indicatorului **Mouse Position** la 100 de pixeli:

```
int nIndex = CommandToIndex(ID_INDICATOR_MOUSE);
m_wndStatusBar.SetPaneInfo(nIndex, ID_INDICATOR_MOUSE,
                           GetPaneStyle(), 100);
```

Observați că stilul casetei de indicare este păstrat cu ajutorul funcției `GetPaneStyle()`, rezultatul acesteia fiind transmis ca al treilea parametru al funcției `SetPaneInfo()`.

Listingul 14.2 folosește toate funcțiile prezentate în cadrul unei funcții exemplificative care se integrează în clasa ferestrei cadru și înscrie un text specificat în interiorul indicatorului **Mouse Position**. Lățimea casetei de indicare este ajustată în funcție de lățimea textului afișat, iar stilul fixat este cel normal.

Textul pentru actualizarea indicatorului din bara de stare cu poziția curentă a mouse-ului poate fi generat de o funcție din clasa reprezentare, așa cum este cea prezentată în listingul 14.3. După asamblare și rulare, proiectul va afișa poziția curentă a mouse-ului, actualizată la deplasarea cursorului (vedeți figura 14.12).

LISTINGUL 14.3 LST15_3.CPP – Generarea unui șir conținând poziția curentă a mouse-ului în scopul afișării într-o casetă de indicare de pe bara de stare

```

1 #include "MainFrm.h"
2 void CStatusMouseView::OnMouseMove(UINT nFlags,
3                                     CPoint point)
4 {
5     // ** Generam un sir continand pozitia mouse-ului
6     CString strMousePosition.Format("(%d, %d)",
7     strMousePosition.Format("(%d, %d)", — ❶
8                                     point.x, point.y);
9
10    // ** Determinam fereastra cadru a aplicatiei SDI
11    CMainFrame* pMainFrame =
12        (CMainFrame*)GetParentFrame();
13
14    // ** Stabilim continutul casetei de indicare
15    pMainFrame->SetMousePosText(strMousePosition); — ❷
16
17    CView::OnMouseMove(nFlags, point);
18 }

```

❶ Poziția mouse-ului este reprezentată sub forma (x,y), în scopul afișării pe bara de stare.

❷ După formatarea poziției mouse-ului în cadrul unui șir de caractere, este apelată funcția care actualizează indicatorul de pe bara de stare, prezentată în listingul 14.2.

Linia 1 din listingul 14.3 include definiția clasei `CMainFrame()`. Funcția de tratare `OnMouseMove()` este apelată ca răspuns la deplasarea mouse-ului și primește o variabilă `point` care conține noua poziție a cursorului mouse-ului. Linia 7 formatează poziția sub forma (x, y), înscriind-o în șirul de caractere `strMousePosition` care a fost declarat în linia 6.

În liniile 11 și 12 se obține un pointer la fereastra cadru care conține obiectul reprezentare curent. Noua funcție membru `SetMousePosText()` a ferestrei cadru (prezentată în listingul 14.2) primește apoi șirul formatat `strMousePosition` în scopul actualizării casetei de indicare.

Dacă implementați această funcție, va trebui să o adăugați împreună cu intrarea corespunzătoare din harta de mesaje, apelând la `ClassWizard` sau la caseta de dialog `New Windows Message and Event Handlers`.

VEDEȚI...

- Pentru mai multe detalii privind tratarea mesajelor generate de deplasarea mouse-ului, vedeți pagina 175.
- Pentru informații referitoare la contextele dispozitiv, vedeți pagina 323.

Despre barele suport în stil Internet Explorer

Bara suport este unul dintre noile controale standard din IE4, având rolul de container pentru bare cu instrumente, bare de dialog și alte controale. În cazul în care folosiți Internet Explorer 4.0, în partea superioară a ferestrei veți găsi un control bară suport care permite glisarea barelor cu instrumente, a barelor de dialog și a casetelor combinate. Zonele de glisare se numesc benzi; un control bară suport poate conține mai multe benzi, acestea prezentându-se sub formă de fâșii orizontale.

Este posibilă, de asemenea, alegerea unor bitmap-uri care să fie afișate pe fundalul barelor suport. Chiar dacă utilizatorul deplasează barele cu instrumente și barele de dialog de-a lungul benzilor unei bare suport, bitmap-ul afișat pe fundal rămâne nealterat.

MFC oferă o clasă de încapsulare pentru o bară suport ancorabilă, numită `CReBar`, care preia de la `CControlBar` funcționalitatea legată de ancorare. Clasa `CReBarCtrl` încapsulează controlul standard bară suport, care implementează o bară suport ancorabilă. Este posibilă obținerea unei referințe la un obiect `CReBarCtrl` pe baza unui obiect `CReBar`, apelând funcția membru `GetReBarCtrl()` a acestuia.

Barele suport reprezintă un subiect vast și complex; din păcate, avem prea puțin spațiu pentru a oferi mai mult de o prezentare sumară a capacităților acestora și a suportului oferit de către infrastructură.

VEDEȚI...

- Pentru a afla despre utilizarea IE, în cadrul unei reprezentări, vedeți pagina 454.

Utilizarea barei suport generată de AppWizard

Dacă doriți să folosiți o bară suport în scopul grupării barelor cu instrumente și a barelor de dialog, puteți instrui `AppWizard` să genereze un control bară suport selectând opțiunea **Internet Explorer Rebars** din caseta de dialog `MFC AppWizard Step 4` (vedeți figura 14.1). În consecință, `AppWizard` va adăuga în definiția clasei `CMainFrame` un obiect de tip `CReBar`, numit `m_wndReBar`, și va genera în funcția `OnCreate()` a ferestrei cadru codul de mai jos pentru crearea și inițializarea barei suport:

```

if (!m_wndReBar.Create(this) ||
    !m_wndReBar.AddBar(&m_wndToolBar) ||
    !m_wndReBar.AddBar(&m_wndDlgBar))
{
    TRACE0("Failed to create rebar\n");
    return -1;    // fail to create
}

```

De unde vin barele suport?

Bara suport este unul din controalele care au apărut odată cu Internet Explorer 4. Denumirea originală este *rebar*, de la *resizable toolbar* (bară dimensionabilă). S-ar putea să întâlniți și termenul *coolbar*.

Funcția `Create()` a clasei `CReBar` primește patru parametri. Primul dintre aceștia este obligatoriu și reprezintă un pointer la fereastra părinte. Prin al doilea parametru puteți să transmiteți o combinație de indicatori de stil (vedeți tabelul 14.6) pentru a specifica stilul barei suport; acest parametru are implicit valoarea `RBS_BANDBORDERS`, efectul fiind trasarea unor margini de încadrare a benzilor. Cel de-al treilea parametru specifică stilurile de fereastră și de ancorare pentru bara suport; în mod implicit sunt considerați indicatorii `WS_CHILD`, `WS_VISIBLE`, `WS_CLIPSIBLINGS` și `CBRS_TOP`, astfel că va fi creată o fereastră copil care este inițial vizibilă și ancorată în partea superioară și care decupează din suprafața sa de desenare barele cu instrumente și barele de dialog conținute. Ultimul parametru specifică un identificator pentru bara suport, acesta fiind implicit `AFX_IDW_REBAR`.

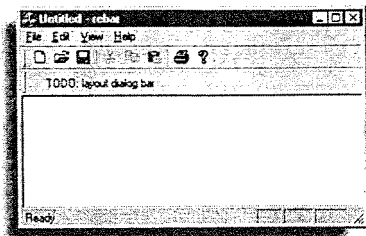
TABELUL 14.6 Indicatori de stil pentru barele suport

Indicator	Descriere
<code>RBS_AUTOSIZE</code>	La modificarea dimensiunii sau poziției barei suport sunt ajustate automat toate benzile componente.
<code>RBS_BANDBORDERS</code>	Trasează margini de separare între benzi.
<code>RBS_FIXEDBORDER</code>	Păstrează aceeași ordine a benzilor.
<code>RBS_VARHEIGHT</code>	În mod normal, toate benzile sunt afișate având aceeași înălțime cu banda cea mai înaltă; dacă transmiteți acest indicator, înălțimea fiecărei benzi va fi exact cea necesară pentru controalele sau barele cu instrumente conținute.
<code>RBS_VERTICALGRIPPER</code>	Bara de agățare este afișată vertical, nu orizontal, în cadrul unei bare suport verticale.
<code>RBS_DBLCLKTOGGLE</code>	Maximizează sau minimizează o bandă a barei suport atunci când utilizatorul efectuează un dublu clic pe aceasta (prin contrast cu un simplu clic care este implicit suficient).
<code>RBS_REGISTERDROP</code>	Permite sesizarea tragerii și plasării OLE prin intermediul unui mesaj <code>RBN_GETOBJECT</code> .

Observați că după crearea barei suport, apelul funcției `AddBar()` a acesteia adaugă în cadrul său o bară cu instrumente standard și o bară de dialog generată de `AppWizard`. Apelurile obișnuite pentru ancorarea barei cu instrumente lipsesc și bara suport își asumă responsabilitatea pentru ancorarea și afișarea ambelor bare de controale pe suprafața sa, așa cum se vede în figura 14.13.

FIGURA 14.13

O aplicație conținând o bară suport standard generată de `AppWizard`.



Bara cu instrumente și bara de dialog pot fi personalizate așa cum am arătat în secțiuni anterioare ale acestui capitol. Bare de controale adiționale pot fi adăugate prin intermediul funcției `AddBar()`.

Mesajele generate de controalele și barele de dialog integrate în barele suport

Mesajele generate de controalele și barele de dialog integrate într-o bară suport sunt trimise către fereastra cadru părinte. Puteți, așadar, să adăugați funcții de tratare pentru aceste controale în oncarea dintre clasele fereastră cadru, reprezentare sau document, așa cum ați procedat și în cazul controalelor de pe barele de dialog.

VEDEȚI...

- Pentru mai multe amănunte privind utilizarea `AppWizard` în scopul generării unei aplicații `SDI`, vedeți pagina 246.
- Pentru a afla despre decupare, vedeți pagina 330.

Stabilirea unui titlu și a unui bitmap de fundal pentru bara suport

Funcția `AddBar()` a obiectului `CReBar` are un singur parametru obligatoriu, și anume un pointer către controlul sau bara cu instrumente care este adăugată. Cel de-al doilea parametru este opțional și vă permite să adăugați sau nu o etichetă alături de bara inserată, transmitând eticheta sub forma unui șir terminat cu caracterul nul sau, respectiv, un pointer `NULL` (aceasta este și valoarea implicită). Puteți, de asemenea, să fixați un bitmap de fundal, transmitând prin intermediul celui de-al treilea parametru un pointer `CBitmap` la un bitmap încărcat în prealabil.

De exemplu, liniile de mai jos specifică o etichetă pentru bara cu instrumente și un bitmap de fundal pentru ambele bare adăugate în bara suport:

```
static CBitmap myBitmap;
myBitmap.LoadBitmap(IDB_MYREBAR_BM);
if (!m_wndReBar.Create(this) ||
    !m_wndReBar.AddBar(&m_wndToolBar, "My Toolbar", &myBitmap) ||
    !m_wndReBar.AddBar(&m_wndDlgBar, NULL, &myBitmap))
```

Observați în prima linie că obiectul `myBitmap` este declarat `static` pentru a rămâne în memorie și după încheierea funcției `OnCreate()`. Ați putea să folosiți în loc o variabilă membru, dar bitmap-ul trebuie să rămână valid pe toată durata de viață a barei suport.

A doua linie încarcă resursa bitmap `IDB_MYREBAR_BM`. Această resursă poate fi creată prin intermediul editorului de resurse bitmap. Apelurile funcției `AddBar()` pentru bara de dialog și bara cu instrumente conțin ambele obiectul `CBitmap` ca al treilea parametru.

La adăugarea obiectului `m_wndToolBar` din linia a patra, prin intermediul celui de-al doilea parametru este transmis un șir de caractere cu rol de etichetă pentru bara cu instrumente. La rularea aplicației, eticheta va fi afișată pe aceeași bandă cu bara cu instrumente, chiar în stânga acesteia. Bara de dialog nu are nici o etichetă, deoarece prin al doilea parametru este transmisă valoarea `NULL`.

VEDEȚI...

- Pentru a afla cum se creează și se utilizează bitmap-urile, vedeți pagina 227.

Despre desenarea în cadrul contextelor dispozitiv

Despre diferitele tipuri de contexte dispozitiv și când trebuie acestea utilizate

Utilizarea și selectarea modurilor de desenare și a capacităților dispozitivelor

Utilizarea modurilor de mapare pentru a desena în inci sau centimetri

Introducere în contextele dispozitiv

Nu vă lăsați descurajat de sintagma *context dispozitiv*. Este una dintre acele denumiri sofisticate care ascunde de fapt o idee simplă și, totodată, un element esențial din Windows. Cum ar arăta Windows dacă ați elimina contextele dispozitiv? N-ar arăta în nici un fel. Un context dispozitiv reprezintă suprafața pe care se desenează toate punctele, liniile, pătratele, fonturile, culorile și tot ceea ce vedeți. Cuvântul „dispozitiv” din *context dispozitiv* înseamnă că puteți să desenați pe ecran, la imprimantă, pe un plotter, într-o cască pentru realitate virtuală sau pe orice alt dispozitiv de afișare în două dimensiuni, fără a cunoaște prea multe detalii despre ce dispozitiv folosiți sau ce marcă sau model este acesta.

Structurile contextelor dispozitiv

În esență, un context dispozitiv este o structură din Windows care conține atribute ce descriu opțiunile implicite de desenare folosite în orice operație grafică, așa cum ar fi trasarea unei linii. Spre deosebire de majoritatea celorlalte structuri din Windows, un program nu poate niciodată să acceseze direct structura unui context dispozitiv, putând doar să modifice opțiunile prin intermediul unui set de funcții de acces standard.

Unul dintre cele mai bune lucruri pe care le-a făcut Microsoft pentru industria de software a fost să standardizeze întreg suportul pentru desenare și dispozitive în cadrul sistemului de operare Windows. Din acel moment, industria de software a crescut exploziv, iar aplicațiile grafice se găsesc din abundență. Acesta este probabil efectul major – și cel mai subapreciat – al sistemului Windows.

Acum pot să mă duc la magazinul de calculatoare din colț, cumpăr cea mai nouă imprimantă XYZ, instalez un singur driver de Windows pus la dispoziție de producător și absolut fiecare aplicație va tipări la acea imprimantă fără ezitare. Contextul dispozitiv este ceea ce le unește și oferă întreg suportul pentru desenare.

VEDEȚI...

- În legătură cu desenarea în cadrul contextelor dispozitiv, vedeți pagina 355.
- În legătură cu utilizarea contextelor dispozitiv pentru imprimante, vedeți pagina 510.

Bibliotecile grafice din Windows

Întreg codul din sistemul de operare care efectuează operații grafice este implementat prin interacțiunea dintre două biblioteci DLL. Prima este întotdeauna GDI.DLL, care „lucrează” cu aplicațiile pentru a oferi o serie de funcții de desenare independente de dispozitiv. A doua bibliotecă este cea dependentă de dispozitiv, putând fi specifică ecranului, imprimantei sau plotterului folosit. Spre exemplu, atunci când desenați pe ecranul unui monitor VGA, biblioteca VGA.DLL este cea care conține codul specific dispozitivului pentru implementarea funcțiilor de desenare.

Tipuri de contexte dispozitiv

Există un context dispozitiv standard brut, și există contexte dispozitiv pentru situații speciale și operații particulare. Contextele dispozitiv efective sunt obiecte GDI (Graphics Device Interface – interfață cu dispozitivele grafice). *GDI* reprezintă o mulțime de funcții care se află într-o bibliotecă DLL din inima sistemului Windows. Aceste funcții fac

legătura dintre apelurile funcțiilor de desenare pe care le efectuați și driverele de dispozitiv care comunică cu dispozitivele hardware pentru a transforma totul în realitate, sub formă de lumină sau cerneală.

Biblioteca de clase de bază Microsoft (MFC) oferă încapsulări ale contextelor dispozitiv care simplifică interacțiunea cu obiectele GDI aflate dedesubt. Clasa care încapsulează contextul dispozitiv standard brut este `CDC`. Această clasă conține un număr imens de funcții de desenare, de mapare de coordonate și de decupare pentru implementarea reprezentărilor grafice. Toate celelalte clase de context dispozitiv, mai specializate, sunt bazate pe această clasă și o extind. Secțiunea următoare prezintă ceva mai amănunțit fiecare dintre aceste clase.

Utilizarea clasei `CDC`

Clasele de context dispozitiv pe care le folosiți au întotdeauna clasa de bază `CDC`. Această clasă conține doi indicatori legați de obiectul GDI aflat dedesubt: indicatorul `m_hDC` și indicatorul `m_hAttribDC`. `m_hDC` este indicatorul de context dispozitiv care face legătura dintre clasă și obiectul GDI pentru a gestiona ieșirile funcțiilor de desenare. `m_hAttribDC` este indicatorul de context dispozitiv care leagă toate informațiile de atribute, cum ar fi culori și moduri de desenare. Nu trebuie să vă preocupați prea mult de acești membri, dar rețineți că la apelarea unor funcții precum `GetDC()` și `ReleaseDC()` (care obțin și eliberează un context dispozitiv pentru o fereastră), acești indicatori GDI sunt atașați și detașați de clasa `CDC`.

Funcțiile de desenare din `CDC`

Clasa `CDC` implementează toate funcțiile de desenare disponibile în Windows. Fiecare dintre aceste implementări nu face decât să apeleze funcțiile de desenare de nivel scăzut corespunzătoare din Win32 (în `GDI.DLL`), transmitând indicatorii de context dispozitiv încapsulați în obiectul `CDC` (`m_hDC` și `m_hAttribDC`).

Capacitatea clasei `CDC` de a se atașa și a desena într-un context dispozitiv poate fi ilustrată printr-un program foarte simplu. Fiecare fereastră are asociat un context dispozitiv care acoperă întreaga fereastră; nu face excepție nici fereastra suprafeței de lucru, care se întinde pe întregul ecran; pentru a vă demonstra puterea și ușurința de utilizare a contextelor dispozitiv, o să fiu inconștient și o să acaparez contextul dispozitiv (`DC`) al suprafeței de lucru pentru a desena direct în cadrul său. Singurul efect negativ care poate apărea este o alterare temporară a ecranului Windows, acesta putând fi regenerat la încheierea programului.

Secvența de pași care urmează are ca efect crearea unei aplicații de tip dialog care va acapara un context dispozitiv.

Crearea unei aplicații de tip dialog care va acapara un context dispozitiv

1. Efectuați un clic pe meniu **File** și selectați **New**.
2. În cadrul casetei de dialog **New**, selectați pagina **Projects** pentru a afișa diferitele tipuri de proiecte.

Fișierul `ReadMe.txt` generat de AppWizard

AppWizard generează un fișier `ReadMe.txt` în directorul noului proiect. Acest fișier text prezintă fiecare dintre fișierele sursă create, împreună cu rolul acestora în cadrul aplicației.

3. Selectați **MFC AppWizard** și introduceți `DCDraw` în caseta **Project Name**.
4. Efectuați un clic pe **OK** pentru a apela caseta de dialog **MFC AppWizard – Step 1**.
5. Selectați tipul de aplicație **Dialog Based Application**.
6. Efectuați un clic pe butonul **Finish** pentru a vedea specificațiile scheletului de proiect.
7. Efectuați un clic pe **OK** și AppWizard va genera fișierele proiectului.

Acum ar trebui să aveți scheletul unei aplicații de tip dialog. Să adăugăm un buton simplu și o funcție de tratare corespunzătoare care va implementa acaparea contextului dispozitiv.

Crearea unui buton

1. Efectuați un clic pe pagina **ResourceView** din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus pentru a deschide resursele proiectului `DCDraw`.
3. Efectuați un clic pe semnul plus din dreptul catalogului **Dialog** pentru a afișa resursele de tip machetă de dialog.
4. Efectuați un dublu clic pe `IDD_DCDRAW_DIALOG` pentru a edita caseta de dialog principală a aplicației.
5. Efectuați un clic pe eticheta **TODO: Place Dialog Controls Here** din macheta de dialog și apăsați **Delete** pentru a o înlătura.
6. Selectați un buton din paleta **Controls**, plasați-l pe caseta de dialog și măriți-l suficient.
7. Apăsați **Alt+Enter** pentru a accesa proprietățile butonului adăugat și introduceți **Draw!** în caseta **Caption**.
8. Înlocuiți identificatorul butonului cu `IDC_DRAWIT`.
9. Apăsați **Enter** pentru a închide caseta de dialog **Push Button Properties**.

Acum ar trebui să aveți un buton pe caseta de dialog, așa că puteți să adăugați o funcție care să răspundă la efectuarea unui clic.

Adăugarea unei funcții de tratare a mesajului `BN_CLICKED` cu ClassWizard

Scurtătură pentru adăugarea unei funcții de tratare a mesajului `BN_CLICKED`

O metodă mai simplă pentru adăugarea funcției de tratare a clicului este să efectuați în cadrul editorului de resurse un dublu clic pe controlul pentru care doriți să creați o funcție de tratare. În acest fel se creează aceeași intrare în harta de mesaje și același cod în funcția de tratare, dar sunt evitați pașii legați de ClassWizard, ajungând direct la pasul 5 și apoi la editarea codului.

1. Efectuați un clic pe noul buton din caseta de dialog (**Draw!** în exemplul nostru) pentru a-l selecta.

2. Apăsați Ctrl+W pentru a apela MFC ClassWizard.
3. Ar trebui să vă aflați în pagina Message Maps a casetei de dialog MFC ClassWizard. Noul butonul `IDC_DRAWIT` ar trebui să fie selectat. În caz că nu este, selectați acest identificator în lista **Object ID**.
4. Efectuați un dublu clic pe mesajul `BN_CLICKED` din lista **Messages** pentru a adăuga o nouă funcție de tratare.
5. Efectuați un clic pe **OK** în caseta de dialog Add Member Function pentru a crea funcția de tratare, acceptând numele implicit `OnDrawit`.
6. Efectuați un clic pe butonul **Edit Code** pentru a edita funcția de tratare a clicului asupra butonului.

Acum puteți implementa în funcția de tratare codul care se ocupă de acaparea contextului dispozitiv și de desenare, așa cum este ilustrat în listingul 15.1.

LISTINGUL 15.1 LST16_1.CPP – Desenarea în contextul dispozitiv al suprafeței de lucru

```
1 void CDCDrawDlg::OnDrawit()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Obținem un pointer la fereastra suprafeței de lucru
6     CWnd* pDesktop = GetDesktopWindow(); — ❶
7
8     // ** Obținem un pointer la contextul dispozitiv al acesteia
9     CDC* pDC = pDesktop->GetWindowDC(); — ❷
10
11     // ** parcurgem 300 de pixeli pe orizontala
12     for(int x=0;x<300;x++)
13     {
14         // ** parcurgem 300 de pixeli pe verticala
15         for(int y=0;y<300;y++)
16         {
17             // ** desenăm fiecare pixel cu alta culoare
18             pDC->SetPixel(x,y,x*y); — ❸
19         }
20     }
21     pDesktop->ReleaseDC(pDC);
22 }
```

- ❶ Această linie determină fereastra suprafeței de lucru (adică întregul ecran).
- ❷ Această linie obține contextul dispozitiv principal pentru fereastra suprafeței de lucru.
- ❸ Fiecare pixel din dreptunghi este desenat cu o culoare calculată pe baza coordonatelor.

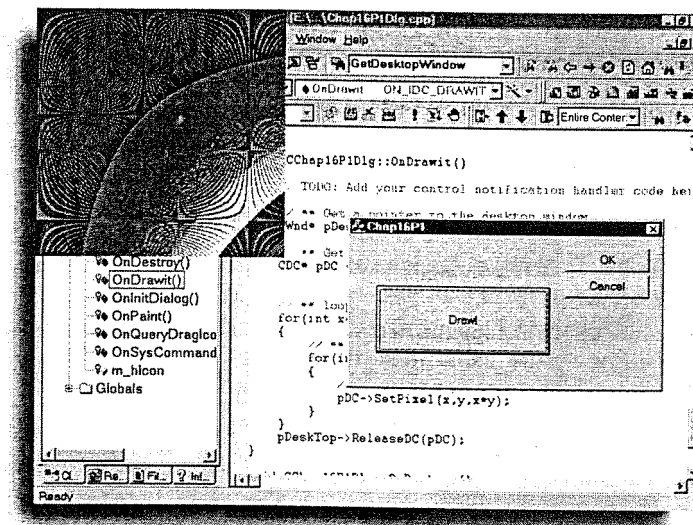
În listingul de mai sus am obținut un pointer la fereastra suprafeței de lucru, apelând funcția `GetDesktopWindow()` în linia 6. Pe baza acestui pointer `CWnd` pot să acaparez contextul dispozitiv principal al ferestrei, folosind funcția `GetWindowDC()` așa cum se vede în linia 9. Liniile de la 12 la 20 sunt responsabile pentru opera de artă înfățișată în figura 15.1. Liniile de la 9 la 12 parcurg 300 de pixeli pe orizontală și 300 de pixeli pe verticală în două bucle `for` de contoare `x` și `y`. Funcția `SetPixel()` este apelată pentru fiecare pixel, primii doi parametri specificând coordonatele pe orizontală și pe verticală (cu 0,0 în colțul stânga sus). Al treilea parametru reprezintă culoarea, exprimată ca o referință de culoare (despre care vom discuta pe larg în capitolul 16, „Utilizarea penițelor și a pensulelor”). Culoarea este aleasă ca produsul dintre coordonatele `x` și `y`, ceea ce duce la surprinzător de interesantul și coloratul efect de *interferență moiré*. În fine, trebuie să eliberăm contextul dispozitiv apelând funcția `ReleaseDC()`, ca în linia 21.

Interferența moiré

Interferența moiré este o figură de interferență creată prin suprapunerea a două modele similare care au secțiuni transparente. Aceasta a fost observată prima dată la mătasea fină moiré, de unde și numele său. Un efect similar poate fi observat la perdelele de tip plasă, unde un strat de material este ușor rotit față de altul. Un model asemănător de interferență stă la baza holografiei.

FIGURA 15.1

Acaparea contextului dispozitiv al suprafeței de lucru și desenarea în cadrul acestuia.



Pentru că `CDCDraw` este atât de neglijentă și desenează direct pe suprafața de lucru, trebuie să reparați și să îndreptați lucrurile. Ați putea să interceptați mesajul `WM_DESTROY` trimis de Windows printr-o funcție `OnDestroy()`. Această funcție va fi apelată la distrugerea casetei de dialog; ar fi, așadar, un loc bun pentru plasarea codului care se ocupă de curățenie.

Adăugarea unei funcții de tratare pentru mesajul `WM_DESTROY`

1. Selectați pagina ClassView din secțiunea spațiului de lucru al proiectului.

2. Deschideți lista claselor din DCDraw efectuând un clic pe semnul plus.
3. Efectuați un clic cu butonul drept pe clasa CDCDrawDlg pentru a apela meniul de context.
4. Selectați din meniu opțiunea **Add Windows Message Handler**.
5. Selectați mesajul **WM_DESTROY** din lista **New Windows Messages/Events**.
6. Efectuați un clic pe butonul **Add and Edit** pentru a adăuga această funcție de tratare în clasa CDCDrawDlg și a începe să o editați.

Acum puteți să adăugați în funcția `OnDestroy()` codul responsabil de curățenie. În acest sens veți solicita suprafeței de lucru să se redeseze și o veți instrui să solicite același lucru ferestrelor sale copil (aplicațiile care rulează). Codul corespunzător este prezentat în listingul 15.2. În linia 8, funcția membru `RedrawWindow()` a ferestrei suprafeței de lucru (care este determinată apelând `GetDesktopWindow()`) forțează redesenarea acesteia. `RedrawWindow()` primește trei parametri; primii doi sunt dreptunghiul și regiunea care trebuie redeseate. Transmițând `NULL` pentru ambii, `RedrawWindow()` presupune în mod adecvat că doriți să fie redesenată întreaga suprafață. Cel de-al treilea parametru este o combinație de indicatori: sunt transmiși `RDW_ERASE` (pentru a șterge totul), `RDW_INVALIDATE` (pentru a determina ulterior redesenarea), `RDW_ALLCHILDREN` (pentru a determina redesenarea tuturor aplicațiilor copil) și `RDW_ERASENOW` (pentru ca totul să se întâmple imediat).

LISTINGUL 15.2 LST16_2.CPP – Redesenarea suprafeței de lucru și a ferestrelor tuturor aplicațiilor cu `RedrawWindow()`

```
1 void CDCDrawDlg::OnDestroy()
2 {
3     CDialog::OnDestroy();
4
5     // TODO: Add your message handler code here
6
7     // ** Redesenam suprafata de lucru si toate ferestrele sale
8     // copil
9     GetDesktopWindow()->RedrawWindow(NULL, NULL, — ❶
10         RDW_ERASE+RDW_INVALIDATE+
11         RDW_ALLCHILDREN+RDW_ERASENOW);
12 }
```

❶ Forțăm redesenarea întregii suprafețe de lucru din Windows și a ferestrelor tuturor aplicațiilor.

Dacă asamblați și rulați programul după efectuarea acestor modificări, ar trebui să obțineți un dreptunghi colorat asemeni celui din figura 16.1. Firește că există o mulțime de funcții de desenare în clasa contextului dispozitiv, toate mai sofisticate decât `SetPixel()`. Dar acestea sunt prea multe și prea complexe pentru a le menționa aici, fiind discutate amănunțit în capitolul 16, „Utilizarea penițelor și a pensulelor” și în capitolul 17, „Utilizarea fonturilor”.

VEDEȚI...

- Pentru desenarea cu penițe și pensule, vedeți pagina 355.
- Pentru afișarea de text în cadrul unui context dispozitiv, vedeți pagina 379.

Utilizarea contextelor dispozitiv client

Clasa `CClientDC` automatizează pentru dumneavoastră apelurile `GetDC()` și `ReleaseDC()`. La construirea unui obiect `CClientDC` veți transmite ca parametru un pointer la o fereastră, iar clasa va folosi apoi acest pointer pentru a accesa contextul dispozitiv al ferestrei respective. La distrugerea clasei, aceasta apelează automat `ReleaseDC()`, așa că nu trebuie să vă mai preocupați. Client se referă la suprafața normală de desenare a ferestrei; există o clasă corespunzătoare, `CWindowDC`, care vă permite să accesați și bara de titlu și marginile ferestrei, dar aceasta este folosită rar pentru că majoritatea aplicațiilor civilizate se rezumă la a desena în propriile zone client.

Accesarea indicatorului de fereastră asociat contextului dispozitiv client

Ați putea avea nevoie să accesați fereastra asociată unui context dispozitiv client. Clasa `CClientDC` reține indicatorul către fereastra asociată în variabila sa membru `m_hWnd`. Puteți să transmiteți indicatorul de fereastră `m_hWnd` (de tip `HWND`) funcției `Attach()` a unui obiect `CWnd` pentru a accesa funcțiile ferestrei asociate.

Puteți modifica exemplul de mai devreme pentru a utiliza un obiect `CClientDC` în locul unui obiect `CDC`. Efectuați un dublu clic pe funcția `OnDrawit()`, afișată în pagina `ClassView` a secțiunii spațiului de lucru al proiectului, și fereastra de editare revine la codul de tratare a butonului.

Modificați funcția `OnDrawit()` în concordanță cu listingul 15.3. În loc să folosiți fereastra suprafeței de lucru, veți desena în contextul dispozitiv al casetei de dialog. În acest scop, obiectul `dlgDC` de tip `CClientDC` primește pointerul `this` (un pointer la clasa curentă), care indică fereastra casetei de dialog. Cum caseta de dialog este o formă de `CWnd`, totul este în regulă și obiectul `dlgDC` conține acum un indicator către propriul context dispozitiv al casetei de dialog. Desenarea se efectuează ca mai înainte, între liniile 9 și 17, cu o mică ajustare; în linia 15, `dlgDC` presupune un apel de tip obiect la `SetPixel()`, și nu un apel de tip pointer. În fine, observați că `ReleaseDC()` nu mai este apelată la sfârșitul funcției `OnDrawit()`; în schimb, indicatorul către contextul dispozitiv va fi eliberat automat de funcția destructor a clasei `CClientDC` în momentul în care funcția se încheie și obiectul `dlgDC` este distrus.

LISTINGUL 15.3 LST16_3.CPP – Utilizarea clasei `CClientDC` pentru accesarea contextului dispozitiv al casetei de dialog

```
1 void CDCDrawDlg::OnDrawit()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Construim un DC client pentru fereastra de dialog
6     CClientDC dlgDC(this); — ❶
7
8     // parcurgem 300 de pixeli pe orizontala
9     for(int x=0; x<300; x++)
```

(continuare)

LISTINGUL 15.3 Continuare

```

10  {
11      // parcurgem 300 de pixeli pe verticala
12      for(int y=0;y<300;y++)
13      {
14          // ** desenam fiecare pixel cu alta culoare
15          pDC->SetPixel(x,y,x*y);
16      }
17  }
18  }

```

❶ Construim un context dispozitiv pentru a accesa în mod corespunzător zona client a casetei de dialog.

Dacă asamblați și rulați aplicația din nou după efectuarea acestor modificări, ar trebui ca la apăsarea butonului **DrawIt** să se deseneze numai în contextul dispozitiv al casetei de dialog, așa cum se vede în figura 15.2.

Forțarea redesenării unei ferestre

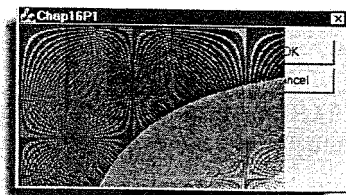
În mod normal, mesajele WM_PAINT sunt trimise numai atunci când o anumită zonă a unei ferestre necesită redesenare. Puteți, însă, să apelați funcția `RedrawWindow()` a clasei `CWnd` pentru a forța trimiterea către respectiva fereastră a unui mesaj WM_PAINT. Prin parametrii transmiși acestei funcții puteți specifica zona care va fi redesenată și dacă fundalul ferestrei va fi și el șters.

Utilizarea contextelor dispozitiv de redesenare

Clasa `CPaintDC` este o încapsulare specială de context dispozitiv care ajută la tratarea mesajului WM_PAINT transmis de Windows. Mesajul WM_PAINT este transmis unei ferestre atunci când o suprafața acesteia a fost descoperită parțial sau total de o altă fereastră; aplicația este informată că regiunea descoperită trebuie redesenată.

FIGURA 15.2

Capturarea contextului dispozitiv al casetei de dialog cu `CClientDC`.



În loc să redesenați întreaga fereastră de fiecare dată când este descoperită o mică porțiune, Windows vă transmite coordonatele unui dreptunghi care încadrează zona descoperită. Puteți să folosiți aceste informații pentru a redesea exclusiv porțiunea afectată, fără a mai irosi timp prețios de procesor pentru a desena lucruri pe care utilizatorul oricum nu poate să le vadă. Nu este obligatoriu să țineți cont de acest dreptunghi; dacă încercați să desenați în afara zonei modificate nu se va întâmpla nimic, deoarece contextul dispozitiv este configurat să decupeze zona de desenare la dreptunghiul în discuție. Dar nu este înțelept

nici să vă bizuiți pe această decupare, deoarece codul de desenare se execută oricum și încetinește astfel totul. În unele cazuri nu aveți ce face – atunci când desenați text și figuri complexe, este prea dificil să determinați unde trebuie decupate textul sau figurile. În acest caz puteți să lăsați *decuparea* în sarcina sistemului Windows. Pixelii pe care i-am desenat reprezintă genul de situație în care pot fi utilizate informațiile pentru redesenare.

Puteți modifica programul de mai devreme pentru a efectua desenarea în cadrul funcției de tratare `OnPaint()`, la primirea unui mesaj de redesenare, ocupându-vă exclusiv de regiunea descoperită. Efectuați un dublu clic pe funcția `OnPaint()` din pagina `ClassView` a secțiunii spațiului de lucru al proiectului. Aceasta vă conduce direct la funcția `OnPaint()`. Remarcați că nu trebuie să creați funcția cu ajutorul `ClassWizard`; `AppWizard` a generat automat codul corespunzător atunci când a creat infrastructura aplicației. Listingul 15.4 prezintă întreaga funcție `OnPaint()`. Linia 24 este linia în care trebuie să adăugați apelul funcției `OnDrawit()`.

Listingul înfățișează alte câteva lucruri interesante create de `AppWizard`. În linia 3 este apelată funcția `IsIconic()`, aceasta întorcând `TRUE` în cazul în care fereastra este minimizată (și este redusă pe bara de sarcini). În acest situație, pictograma aplicației este afișată în centrul dreptunghiului care încadrează fereastra minimizată, în linia 18. Ramura `FALSE` a testului `IsIconic()` poate fi utilizată pentru a implementa propriul cod de redesenare, motiv pentru care în linia 24 este apelată funcția `OnDrawit()`.

LISTINGUL 15.4 LST16_4.CPP – Modificarea funcției de tratare `OnPaint()` a aplicației de tip dialog

```

1 void CDCDrawDlg::OnPaint()
2 {
3     if (IsIconic())
4     {
5         CPaintDC dc(this); // device context for painting
6
7         SendMessage(WM_ICONERASEBKGND, ❶
8             (WPARAM) dc.GetSafeHdc(), 0);
9
10        // Center icon in client rectangle
11        int cxIcon = GetSystemMetrics(SM_CXICON);
12        int cyIcon = GetSystemMetrics(SM_CYICON);
13        CRect rect;
14        GetClientRect(&rect);
15        int x = (rect.Width() - cxIcon + 1) / 2;
16        int y = (rect.Height() - cyIcon + 1) / 2;
17
18        // Draw the icon
19        dc.DrawIcon(x, y, m_hIcon); ❷
20    }

```

(continuare)

LISTINGUL 15.4 Continuare

```

21     else
22     {
23         // ** Apelam propria functie de desenare
24         OnDrawit(); — ❸
25     }
26 }

```

- ❶ Șterge fundalul ferestrei minimizezate la o pictogramă.
 ❷ Desenează pictograma în mijlocul ferestrei minimizezate.
 ❸ Linia adăugată, folosită pentru apelarea funcției de desenare.

Acum trebuie să modificați funcția `OnDrawit()` pentru a utiliza `CPaintDC`. Aceste modificări sunt ilustrate în listingul 15.5. Observați că în linia 6 este folosită clasa `CPaintDC` pentru construirea obiectului `paintDC` asociat ferestrei dialogului. Acesta funcționează corect doar atunci când fereastra răspunde la mesajul `WM_PAINT`. În acest caz, funcția de desenare este apelată din `OnPaint()`, deci totul este în regulă. Cum codul de desenare a rămas în funcția de tratare a butonului, execuția sa se poate face și prin efectuarea unui clic pe butonul `DrawIt`, dar acum nu se va mai întâmpla nimic pentru că fereastra nu mai are regiuni invalidate – ceea ce înseamnă că va fi decupată total la desenare.

Următoarea modificare apare în linia 9, acolo unde este declarat un pointer `RECT` care este inițializat cu adresa dreptunghiului `rcPaint` din membrul `m_ps` al obiectului `paintDC`. Acest dreptunghi reprezintă zona care trebuie redesenată și este folosit în liniile 12 și 15 pentru ca desenarea să fie restrânsă la suprafața sa. Actualizarea va fi astfel accelerată substanțial atunci când nu trebuie redesenată decât o mică porțiune a ferestrei. `SetPixel()` consumă destul de mult timp atunci când este apelat de multe ori; acoperirea zonei de 300x300 pixeli necesită apelarea `SetPixel()` de 90.000 de ori! În cazul în care zona descoperită nu are decât 50x30 pixeli, va trebui să apălați `SetPixel()` de doar 1.500 de ori, ceea ce înseamnă o actualizare a ferestrei de 60 de ori mai rapidă – merită efortul.

Observați o altă modificare: `SetPixel()` este apelată acum pe baza obiectului `paintDC`, în linia 18. `SetPixel()` este implementată în clasa `CDC`, deci toate funcțiile de desenare rămân disponibile în oricare dintre clasele context dispozitiv specializate prin derivare.

LISTINGUL 15.5 LST16_5.CPP – Utilizarea clasei `CPaintDC` pentru tratarea cererilor `WM_PAINT`

```

1 void CDCDrawDlg::OnDrawit()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Construim un DC de redesenare pentru fereastra dialog
6     CPaintDC paintDC(this)
7

```

```

8     // ** Cream un pointer scurtatura catre dreptunghi
9     RECT* pRect = &paintDC.m_ps.rcPaint; — ❶
10
11     // ** parcurgem dreptunghiul de desenare pe horizontala
12     for(int x=pRect->left; x < pRect->right ;x++)
13     {
14         // ** parcurgem dreptunghiul de desenare pe verticala
15         for(int y=pRect->top; y < pRect->bottom ;y++)
16         {
17             // ** desenam fiecare pixel cu alta culoare
18             paintDC.SetPixel(x,y,x*y);
19         }
20     }
21 }

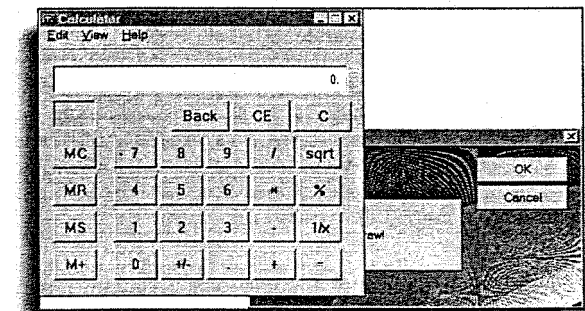
```

- ❶ Observați că de această dată se desenează numai în dreptunghiul de redesenare.

Asamblați și rulați aplicația după efectuarea acestor modificări și veți vedea comportamentul contextului dispozitiv de redesenare. Mai întâi ar trebui să vedeți caseta de dialog având fundalul colorat, dar cu butoanele vizibile; la prima afișare a ferestrei este transmis un mesaj `WM_PAINT` pentru desenarea întregii suprafețe, după care sunt afișate butoanele, iar acestea se suprapun peste zona desenată. Pentru a vedea efectul mesajului de redesenare, acoperiți o jumătate din caseta de dialog cu o altă fereastră (Windows Calculator este foarte nimerită). Aplicația Calculator poate fi lansată prin intermediul butonului **Start**; selectați **Programs, Accessories** și apoi selectați **Calculator**. Dispunerea ferestrelor este înfățișată în figura 15.3.

FIGURA 15.3

Experimentarea mesajului `WM_PAINT` prin descoperirea ferestrei.



După ce ați plasat calculatorul astfel încât să acopere jumătate din caseta de dialog `DCDraw`, efectuați un clic pe suprafața acesteia pentru a o aduce în prim-plan. Veți vedea că zona descoperită este redesenată. Operația de redesenare necesită câteva momente – o jumătate de secundă pe vechiul meu P166. Acum selectați din nou calculatorul și trageți-l puțin în stânga și în sus. Eliberați-l și zona descoperită din caseta de dialog ar trebui să fie redesenată ceva mai repede. Continuați astfel și veți observa cum caseta de dialog este redesenată din ce în ce mai rapid. Având acoperită de calculator o mică porțiune din colțul

stânga sus al casetei de dialog, selectați-o din nou și remarcați timpul necesar redesenării – mult mai scurt. Aceasta se întâmplă pentru că regiunea descoperită este destul de mică și programul nu redesenează decât acest dreptunghi invalid. Sunt necesare mai puține apeluri `SetPixel()`, așa că totul este desenat mult mai repede. Puteți să deplasați cele două ferestre una peste cealaltă pentru a sesiza efectul mesajului de redesenare. Toate aplicațiile Windows trebuie să răspundă la acest mesaj, chiar dacă de multe ori este ascuns într-o clasă control sau reprezentare pentru a nu vă da bătăi de cap.

Poate vă amintiți membrul `m_ps` al clasei `CPaintDC`. Acesta este o structură `PAINTSTRUCTURE` care conține dreptunghiul `rcPaint` pe care l-ați folosit pentru a determina zona ce trebuie redesenată. Există, însă, și alți membri utili.

Structura `PAINTSTRUCT` este definită astfel:

```
typedef struct tagPAINTSTRUCT {
    HDC hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[16];
} PAINTSTRUCT;
```

Membrul `hdc` este indicatorul către contextul dispozitiv GDI aflat la bază. Există un indicator de condiție, `fErase`, pe care Windows îl fixează la `TRUE` dacă este necesară ștergerea fundalului înainte de desenare. Membrul `rcPaint` a fost utilizat în listingul 15.5 pentru determinarea dreptunghiului invalid care trebuie redesenat. Ultimii trei membri sunt declarați ca *rezervați* de către Windows în documentația Microsoft, așa că este mai bine probabil să-i lăsați deoparte.

Tratarea mesajului WM_ERASEBKGD

O altă soluție pentru a determina dacă este necesară ștergerea fundalului este să adăugați o funcție de tratare a mesajului `WM_ERASEBKGD`. Noua funcție de tratare, `OnEraseBkGnd()`, va fi apelată de fiecare dată când este necesară ștergerea fundalului. Funcția `OnEraseBkGnd()` primește ca prim parametru un pointer la obiectul context dispozitiv în cauză (un obiect `CDC`).

Poate vă întrebați de unde sunt adunate toate aceste informații în cadrul obiectului `CPaintDC`. Răspunsul se află într-o funcție a clasei `CWnd`, numită `BeginPaint()`. Vă amintesc că clasa `CDialog` este derivată și afișată, asemeni multor altor controale, din `CWnd`. Prin urmare, în aceste clase derivate este disponibilă întreaga funcționalitate de bază a ferestrelor și toate folosesc `BeginPaint()` pentru a inițializa structura `PAINTSTRUCT` la declararea unui obiect `CPaintDC`. Atunci când obiectul `CPaintDC` este distrus, se apelează o funcție corespunzătoare a clasei `CWnd`, numită `EndPaint()`, care informează Windows că mesajul `WM_PAINT` a fost tratat și că operația de redesenare a regiunii afectate din fereastră a fost efectuată.

Utilizarea contextelor dispozitiv de memorie

Un *context dispozitiv de memorie* este un context dispozitiv care nu are asociat nici un dispozitiv. La început ați putea crede că este ceva ciudat, dar veți vedea că este foarte util. Aceste contexte dispozitiv se folosesc de regulă împreună cu un context dispozitiv obișnuit pentru copierea și lipirea unor zone de ecran. Este posibilă crearea unui context dispozitiv de memorie compatibil cu un context dispozitiv de afișare. Apoi puteți să copiați în memorie imaginile care nu mai sunt afișate, aducându-le înapoi în contextul dispozitiv de afișare atunci când este nevoie.

Nu există nici o clasă MFC care să încapsuleze un context dispozitiv de memorie; așa ceva nu este necesar deoarece clasa `CDC` este foarte potrivită. Fiind vorba despre o clasă context dispozitiv `CDC`, o puteți folosi pentru desenare ca de obicei; singura diferență este că veți desena în memorie, iar utilizatorul nu va vedea nimic până când nu copiați rezultatele în contextul dispozitiv de afișare al unei ferestre.

Puteți să creați un context dispozitiv de memorie compatibil și să modificați codul de desenare pentru a lucra cu memoria și nu cu ecranul. Odată desenată imaginea în memorie, puteți apela funcția `BitBlt()` pentru a o copia pe ecran. Modificările ilustrate de listingul 15.6 sunt relativ majore, deoarece trebuie să creați nu doar un context dispozitiv de memorie, ci și un bitmap asociat acestuia. Spre deosebire de contextele dispozitiv de afișare, care sunt legate de ferestre, un context dispozitiv de memorie nu dispune automat de un bitmap compatibil cu ecranul.

Dimensiunile bitmap-urilor în contextele dispozitive de memorie

Un bitmap dintr-un context dispozitiv de memorie poate fi mult mai mic sau mult mai mare decât contextul dispozitiv de afișare asociat. Prin crearea unui bitmap mare, puteți obține o suprafață de desenare cu o rezoluție mai înaltă decât cea a ecranului. Trebuie să rețineți, însă, că un bitmap mai mare are nevoie de mai multă memorie de la sistem. În cazul în care bitmap-ul necesită mai mult decât memoria RAM existentă, sistemul va fi încetinit considerabil prin utilizarea memoriei virtuale în scopul compensării lipsei de RAM. Memoria virtuală este înecată din cauză că segmente mari de memorie, numite pagini, sunt schimbate tot timpul între RAM și disc.

Dispozitivul context de redesenare a redevenit un context dispozitiv client, iar contextul dispozitiv de memorie este declarat în linia 9 a listingului 15.6 sub forma obiectului `memDC`. În linia 10 este apelată funcția `CreateCompatibleDC()` a contextului dispozitiv. Funcția primește obiectul `clientDC`, ghidându-se după el pentru a stabili dimensiunea și celelalte atribute ale contextului dispozitiv de memorie care-l fac pe acesta compatibil cu fereastra de pe ecran.

În linia 17 este declarat un bitmap care va fi utilizat pentru desenare în cadrul contextului dispozitiv de memorie. Acest bitmap este creat prin apelul funcției membru `CreateCompatibleBitmap()` din linia 18. În apelul funcției `CreateCompatibleBitmap()` este transmisă variabila `clientDC`, indicând tipul de context dispozitiv împreună cu care va fi utilizat bitmap-ul; acesta este folosit ca referință în determinarea numărului de culori pe care le va suporta bitmap-ul. Lățimea și înălțimea zonei client sunt determinate în linia 14, fiind transmise ca parametri pentru a indica

dimensiunea bitmap-ului. În linia 22, bitmap-ul este selectat în cadrul contextului dispozitiv prin intermediul funcției membru `SelectObject()`. Orice operație de desenare în contextul dispozitiv de memorie va folosi de acum acest bitmap.

Desenarea modelului se poate face acum ca de obicei; cu toate acestea, desenarea va avea loc în memorie, nu direct pe ecran. În linia 36, apelul `BitBlt()` copiază bitmap-ul înapoi în contextul dispozitiv de afișare. `BitBlt()` este o funcție de copiere a imaginilor, iar numele său ciudat provine de la un termen hardware mai vechi, *bit blitting*, care se referă la un circuit capabil să copieze rapid o zonă de memorie dintr-un loc în altul. În unele cazuri s-ar putea ca placa grafică să efectueze exact acest gen de copiere la apelul funcției, dar în alte cazuri copierea va fi efectuată de procesor. `BitBlt()` copiază din `memDC` în `clientDC`, primind ca parametri lățimea, înălțimea și punctul de început, atât pentru sursă, cât și pentru destinație.

Utilizarea contextelor dispozitiv de memorie compatibile și copierea imaginilor

Dacă vrei să copiezi imagini color de pe ecran într-un context dispozitiv de memorie sau invers, trebuie întotdeauna să creai și să selectezi un bitmap compatibil cu un context dispozitiv de afișare real.

Un parametru interesant transmis acestei funcții este indicatorul `SRCINVERT`, care determină `BitBlt()` să copieze imaginea inversând culorile. În consecință, imaginea din fereastra destinație va avea un colorit diferit, ceea ce demonstrează că desenarea are loc într-adevăr în memorie și imaginea este copiată apoi în contextul dispozitiv al casetei de dialog.

Operați aceste modificări așa cum o arată listingul 15.6 și înlăturați apelul `OnDrawit()` din funcția de tratare `OnHandler()` (altfel veți obține niște efecte foarte ciudate); apoi asamblați și lansați în execuție. La efectuarea unui clic pe butonul **DrawIt** veți putea observa o mică întârziere în timp ce imaginea este desenată în memorie. După aceea, imaginea având culorile inversate este copiată în contextul dispozitiv al casetei de dialog.

LISTINGUL 15.6 LST16_6.CP – Desenarea într-un context dispozitiv de memorie

```
1 void CDCDrawDlg::OnDrawit()
2 {
3     // TODO: Add your control notification handler code
4
5     // ** Construim un DC client pentru fereastra dialog
6     CClientDC clientDC(this);
7
8     // ** Cream un context dispozitiv de memorie compatibil
9     CDC memDC;
10    memDC.CreateCompatibleDC(&clientDC); — ❶
11
12    // ** Determinam zona client
13    CRect rcClient;
14    GetClientRect(&rcClient);
15
```

```
16    // ** Cream un bitmap compatibil
17    CBitmap memBitmap;
18    memBitmap.CreateCompatibleBitmap(&clientDC, — ❷
19        rcClient.Width(), rcClient.Height());
20
21    // ** Il selectam in cadrul contextului dispozitiv de memorie
22    memDC.SelectObject(&memBitmap);
23
24    // ** parcurgem dreptunghiul de desenare pe orizontala
25    for(int x=0; x < rcClient.Width(); x++)
26    {
27        // ** parcurgem dreptunghiul de desenare pe verticala
28        for(int y=0; y < rcClient.Height(); y++)
29        {
30            // ** desenam fiecare pixel cu alta culoare
31            memDC.SetPixel(x,y,x*y); — ❸
32        }
33    }
34
35    // ** Copiem imaginea din memorie inapoi in contextul
36    // dispozitiv client
37    clientDC.BitBlt(0,0, — ❹
38        rcClient.Width(), rcClient.Height(),
39        &memDC, 0,0, SRCINVERT);
40 }
```

❶ Cream un context dispozitiv de memorie care să fie compatibil cu un context dispozitiv de pe ecran.

❷ Bitmap-ul trebuie să fie și el compatibil cu atributele contextului dispozitiv de pe ecran.

❸ Toată desenarea are loc în memorie; în acest moment nu se vede nimic.

❹ Întreaga imagine este copiată acum pe ecran, devenind vizibilă utilizatorului.

Utilizarea modurilor de mapare

Modurile de mapare reprezintă un alt instrument util pe care-l oferă contextele dispozitiv. Un mod de mapare se referă, în esență, la faptul că puteți desena pe ecran sau la imprimantă folosind unități precum inci sau centimetri, în loc de pixeli, obținând o reprezentare la scară destul de rezonabilă.

Atunci când folosiți moduri de mapare, veți întâlni termeni precum *unități logice* și *unități dispozitiv*. Pe scurt, o *unitate dispozitiv* este un pixel, sau este cea mai mică unitate indivizibilă pe care o poate reprezenta dispozitivul. O unitate logică poate fi o unitate de măsură britanică, metrică sau legată de fonturi.

În mod implicit, contextul dispozitiv folosește un mod de mapare numit `MM_TEXT`. Acest nume ciudat semnifică faptul că o unitate logică este egală cu o unitate dispozitiv, astfel că

nu apar conversii și toate coordonatele pe care le precizați într-o comandă de desenare se exprimă în pixeli. Există mai multe moduri de mapare, prezentate în tabelul 15.1.

Spațiul de coordonate și axa verticală

Toate funcțiile care desenează într-un context dispozitiv necesită parametri ce specifică poziții în spațiul de coordonate global. Spațiul de coordonate este o suprafață bidimensională care are două axe perpendiculare, ca în geometria Carteziană. Fiecare punct are o componentă pe verticală și una pe orizontală, acestea fiind transmise ca parametri individuali către funcțiile de desenare.

Atunci când se folosește modul de mapare `MM_TEXT`, valorile mai mici pe verticală se află în partea superioară a ecranului, iar valorile mai mici pe orizontală se află în partea stângă. În schimb, celelalte moduri de mapare plasează valorile mici pe verticală în partea inferioară a ecranului, în timp ce valorile mici pe orizontală rămân în partea stângă.

TABELUL 15.1 Modurile de mapare posibile

Indicator de mapare	0 unitate logică este echivalentă cu
<code>MM_TEXT</code>	Un pixel
<code>MM_LOMETRIC</code>	0.1 dintr-un milimetru
<code>MM_HIMETRIC</code>	0.01 dintr-un milimetru
<code>MM_LOENGLISH</code>	0.01 dintr-un inci
<code>MM_HIENGLISH</code>	0.001 dintr-un inci
<code>MM_TWIPS</code>	1/1440 dintr-un inci, sau 1/20 dintr-un punct (legat de fonturi)
<code>MM_ISOTROPIC</code>	Valoare definită de utilizator, dar întotdeauna egală pe X și pe Y
<code>MM_ANISOTROPIC</code>	Valoare definită de utilizator

Aceste moduri de mapare sunt stabilite prin intermediul unei funcții a contextului dispozitiv, `SetMapMode()`, care primește pur și simplu unul dintre indicatorii enumerați. Odată stabilită maparea, coordonatele pe care le transmiteți în funcțiile de desenare sunt convertite intern de către GDI la unități dispozitiv (sau pixeli), pentru fiecare mod de mapare. Așadar, dacă modul de mapare este `MM_LOMETRIC` și transmiteți valoarea 100 unei funcții de desenare, aceasta va înțelege că vă referiți la 100x0,1 dintr-un milimetru, ceea ce înseamnă un centimetru. Se va calcula apoi numărul de pixeli pe care ecranul sau imprimanta îi pot afișa într-un centimetru și se folosește această conversie.

`MM_ISOTROPIC` și `MM_ANISOTROPIC` vă permit să stabiliți propriile scalări, folosind în acest scop funcțiile `GetWindowExt()` și `SetViewportExt()`. Există, de asemenea, funcțiile `SetWindowOrg()` și `SetViewportOrg()`, care permit modificarea coordonatelor originii (0,0). Un alt lucru care trebuie reținut este că toate modurile de mapare în afară de `MM_TEXT` consideră axa verticală orientată în sus. Este un aspect ce poate deruta atunci când sunteți obișnuiți să lucrați cu coordonate exprimate în pixeli și ați putea descoperi că ceea ce desenați apare cu susul în jos sau depășește marginea superioară a ecranului. Coordonatele negative nu sunt deloc neobișnuite, mai ales atunci când modificați originea ferestrei.

Ați putea să utilizați o parte din aceste cunoștințe într-un mic program, folosind o reprezentare a unei aplicații SDI. Creați o astfel de aplicație urmând secvența de pași „Crearea unei aplicații SDI” din capitolul 12, „Utilizarea documentelor, a reprezentărilor și a cadrelor”. Acceptați toate opțiunile implicite, mai puțin cea din pasul 1, în care trebuie să selectați tipul de aplicație SDI, și denumiți proiectul `MapMode`.

Clasa reprezentare a aplicației `MapMode` are o funcție membru `OnDraw()` pe care o veți folosi de obicei pentru a implementa codul de desenare. Urmăți pașii din secvența următoare pentru a edita funcția `OnDraw()`.

Localizarea și editarea funcției `OnDraw()` a reprezentării

1. Selectați pagina `ClassView` a secțiunii spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus pentru a afișa clasele proiectului.
3. Efectuați un clic pe semnul plus alăturat clasei `MapModeView` pentru a afișa membrii acesteia.
4. Efectuați un dublu clic pe funcția `OnDraw()` a clasei reprezentare pentru a o putea edita.

Acum puteți să adăugați noile linii de cod din funcția `OnDraw()`, ca în listingul 15.7. Pentru început, observați că modul de mapare este stabilit în linia 9 la `MM_LOMETRIC`, ceea ce înseamnă că fiecare unitate de coordonate va reprezenta 0,1mm. În linia 10 se determină dreptunghiul client prin apelul `GetClientRect()`. Oricare ar fi modul de mapare, această funcție vă furnizează întotdeauna coordonate exprimate în pixeli (sau în unități dispozitiv), așa că în linia 16 veți apela funcția `DpToLp()` a contextului dispozitiv pentru a efectua conversia la coordonate logice.

`DpToLp()` vine de la Device Points to Logical Points (puncte dispozitiv la puncte logice) și poate să convertească obiecte `CPoint` sau `CRect` transmise ca pointeri. Există o funcție inversă, `LpToDp()`, care convertește coordonatele logice în coordonate dispozitiv. În bucla din liniile 19-21, variabila `x` parcurge dreptunghiul client de la stânga la dreapta, acesta fiind convertit deja în coordonate logice. Bucla iterează cu câte 100 de unități logice, ceea ce înseamnă un centimetru în maparea `MM_LOMETRIC`. Cele trei apeluri `SetPixel()` trasează pe suprafața ferestrei marcaje aflate la distanță de 1 cm.

LISTINGUL 15.7 `LST16_7.CPP` – Utilizarea modului de mapare `MM_LOMETRIC`

```

1 void CMapModeView::OnDraw(CDC* pDC)
2 {
3     CMapModeDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // ** Stabilim modul de mapare la unitati de 0,1 mm
9     pDC->SetMapMode(MM_LOMETRIC);
10
```

(continuare)

LISTINGUL 15.7 Continuare

```

11 // ** Determinam zona client (in coordonate dispozitiv)
12 CRect rcClient;
13 GetClientRect(&rcClient);
14
15 // ** Convertim coordonatele dispozitiv in coordonate logice
16 pDC->DPtoLP(&rcClient); — 2
17
18 // ** Iteram cu cate 100 de unitati logice
19 for(int x=rcClient.TopLeft().x;
20     x<rcClient.BottomRight().x;
21     x+=100)
22 {
23     // ** Desenam trei pixeli in jurul coordonatei y
24     pDC->SetPixel(x,rcClient.CenterPoint().y-1,0); — 3
25     pDC->SetPixel(x,rcClient.CenterPoint().y,0);
26     pDC->SetPixel(x,rcClient.CenterPoint().y+1,0);
27 }
28 }

```

1 Aici se stabilește un mod de mapare în care fiecare unitate măsoară 0,1 dintr-un milimetru.

2 DPtoLP convertește coordonatele dispozitiv exprimate în pixeli la coordonate logice (acum 0,1 mm).

3 Cele trei apeluri ale funcției SetPixel() sunt folosite pentru a afișa o mică linie verticală.

Observați că funcția OnDraw() primește un pointer la contextul dispozitiv care este folosit pentru toate operațiile de desenare. Infrastructura aplicației este cea care pregătește în prealabil acest context dispozitiv. În capitolul 22, „Tipărirea și previzualizarea”, veți vedea cum este pregătit acest context dispozitiv pentru ecran sau pentru imprimantă. Dacă asamblați și rulați aplicația, ar trebui să vedeți un rând orizontal de marcate dispuse la intervale regulate. Dacă măsurați cu o riglă, ar trebui să se confirme dispunerea marcajelor la distanță de un centimetru.

Acuratețea mapării la afișarea pe ecran

Nu vă așteptați ca maparea pe ecran să fie perfectă; diferențele existente între dimensiunile monitoarelor și între unitățile dispozitiv virtuale pe verticală și orizontală afectează acuratețea. Tipărirea la imprimantă este mai precisă.

Încercați să înlocuiți indicatorul MM_LOMETRIC cu MM_LOENGLISH. După asamblarea și rularea programului, distanța dintre puncte ar trebui să fie mai mare, măsurând un inci.

Moduri de mapare cu scalare liberă

Este posibilă stabilirea unor rapoarte de mapare proprii, ceea ce netezește calea pentru o implementare ușoară a scalărilor prin intermediul modului de mapare nefericit denumit

MM_ANISOTROPIC. Nefericit denumit pentru că *anizotropic* înseamnă cu proprietăți diferite în direcții diferite; deși este un fapt indubitabil adevărat pentru acest mod de mapare, derutează prin faptul că modul de mapare permite scalări diferite pe cele două axe. Câinele meu este anizotropic, dar nu poate fi scalat atât de ușor pe cât o permite acest mod de mapare. În ciuda denumirii mai ciudate, acest mod de mapare este nemaipomenit de puternic pentru că vă permite să stabiliți exact conversia dintre unitățile logice și unitățile dispozitiv.

Listingul 15.8 ilustrează această conversie, folosind funcția SetViewportExt() pentru a stabili unitățile dispozitiv astfel încât lățimea zonei client să rămână aceeași și înălțimea ei să măsoare 500 de pixeli dispozitiv. Apoi, în linia 19, lățimea logică este fixată la 10.000 de unități prin apelul funcției SetWindowExt(), în timp ce înălțimea rămâne de 500 de pixeli logici. Aceasta înseamnă că fereastra va măsura pe orizontală exact 10.000 de unități logice. Bucla din liniile 23-30 desenează de-a lungul ferestrei exact 100 de marcate, dispuse la câte 100 de unități logice distanță. Pe de altă parte, fiecare unitate logică de pe verticală este egală cu un pixel (ca în MM_TEXT).

Dacă asamblați și rulați aplicația după efectuarea în funcția OnDraw() a modificărilor prezentate în listingul 15.8, ar trebui să vedeți 100 de marcate dispuse regulat de-a lungul ferestrei. Dacă încercați să modificați dimensiunile ferestrei, veți vedea că marcasele sunt distanțate sau apropiate, rămânând întotdeauna în număr de 100 și dispuse regulat.

LISTINGUL 15.8 LST16_8.CPP – Utilizarea modului de mapare MM_ANISOTROPIC

```

1 void CMapModeView::OnDraw(CDC* pDC)
2 {
3     CMapModeDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     pDC->SetMapMode(MM_ANISOTROPIC); — 1
9
10    CRect rcClient;
11    GetClientRect(&rcClient);
12
13    // ** Fixam dimensiunile dispozitivului la latimea zonei
14    // ** client si inaltimea de 500 de pixeli
15    pDC->SetViewportExt(CSize(rcClient.BottomRight(). — 2
16    >=x,500));
17
18    // ** Fixam dimensiunile logice la 10.000 de unitati pe
19    // ** orizontala si 500 de pixeli pe verticala
20    pDC->SetWindowExt(CSize(10000,500)); — 3

```

(continuare)

LISTINGUL 15.8 Continuare

```

21  pDC->DPtoLP(&rcClient);
22
23  for(int x=rcClient.TopLeft().
24      x;x<rcClient.BottomRight().x;
25      x+=100)
26  {
27      pDC->SetPixel(x,rcClient.CenterPoint().y-1,0);
28      pDC->SetPixel(x,rcClient.CenterPoint().y,0);
29      pDC->SetPixel(x,rcClient.CenterPoint().y+1,0);
30  }
31  }

```

- ❶ MM_ANISOTROPIC permite stabilirea de scalări arbitrare pe fiecare dintre axe.
- ❷ Funcția SetViewportExt() stabilește lățimea și înălțimea dispozitivului în pixeli.
- ❸ Funcția SetWindowExt() stabilește lățimea și înălțimea logice ale ferestrei în coordonate logice.

Modul de mapare MM_ISOTROPIC își merită probabil mai bine numele și presupun că de aici își trage denumirea omologul său, MM_ANISOTROPIC. *Izotropic* înseamnă cu aceleași proprietăți pe direcții diferite. Puteți folosi aceleași funcții SetWindowExt() și SetViewportExt() pentru fixarea scalării, dar GDI va ajusta valorile diferite astfel încât o unitate logică pe orizontală să fie întotdeauna egală cu o unitate logică pe verticală. Acest mod este util în cazul în care doriți să păstrați proporția între dimensiunile a ceea ce desenați.

Utilizarea funcțiilor SetWindowExt() și SetViewportExt()

Funcțiile SetWindowExt() și SetViewportExt() sunt folosite exclusiv în modulele de mapare MM_ISOTROPIC și MM_ANISOTROPIC; în celelalte moduri sunt ignorate.

VEDEȚI...

➤ Pentru utilizarea modurilor de mapare la tipărire, vedeți pagina 510.

Determinarea caracteristicilor unui dispozitiv

Un context dispozitiv poate fi interogată direct pentru a determina caracteristicile dispozitivului asociat. În situațiile cele mai uzuale sunt suportate cam toate funcțiile standard de desenare și de afișare a fonturilor. Dar unele dispozitive – plotterele, de exemplu – suportă numai trasarea de linii. Imprimantele pot fi monoculare sau s-ar putea să suporte o mulțime restrânsă de culori. O singură funcție, GetDeviceCaps(), vă spune tot ce ați putea dori să aflați despre un anumit dispozitiv. Așa cum probabil ați intuit, pentru ca această funcție să ofere atât de multe informații trebuie să existe o serie de indicatori care să poată fi transmiși în scopul interogării diferitelor aspecte tehnice.

Funcția membru GetDeviceCaps() a unui context dispozitiv primește un singur parametru, care reprezintă un indicator ce specifică informația dorită. Acești indicatori sunt grupați în diferite mulțimi, în funcție de informațiile corespunzătoare.

Indicatorul TECHNOLOGY interoghează tipul tehnologiei folosite de dispozitivul asociat; poate fi întoarsă oricare dintre valorile prezentate în tabelul 15.2.

TABELUL 15.2 Valorile întoarse de GetDeviceCaps() pentru indicatorul TECHNOLOGY

Valoarea întoarsă	Descriere
DT_RASDISPLAY	Placă grafică și monitor standard
DT_RASPRINTER	Imprimantă bitmap standard
DT_PLOTTER	Dispozitiv plotter cu trasare de linii
DT_RASCAMERA	Dispozitiv de intrare bitmap
DT_CHARSTREAM	Secvență de caractere, așa cum este tastatura
DT_METAFILE	Fișier care conține instrucțiuni pentru trasarea unui desen
DT_DISPPFILE	Fișier de afișare

Versiunea driverului poate fi obținută cu ajutorul indicatorului DRIVERVERSION.

Dimensiunile dispozitivului pot fi determinate prin intermediul mai multor indicatori, prezentați în tabelul 15.3.

TABELUL 15.3 Indicatorii GetDeviceCaps() pentru determinarea dimensiunilor fizice

Indicatorul transmis	Descrierea valorii întoarse
HORZRES	Lățimea în pixeli a dispozitivului
VERTRES	Înălțimea în pixeli a dispozitivului
HORZSIZE	Lățimea în milimetri a dispozitivului
VERTSIZE	Înălțimea în milimetri a dispozitivului
ASPECTX	Înălțimea în pixeli a dispozitivului, relativ la lățime
ASPECTY	Lățimea în pixeli a dispozitivului, relativ la înălțime
ASPECTXY	Lungimea în pixeli a diagonalei dispozitivului

Caracteristicile legate de desenare

Există o serie întreagă de indicatori care furnizează caracteristicile privind trasarea curbelor (CURVECAPS), trasarea liniilor (LINECAPS), trasarea poligoanelor (POLYGONCAPS) și afișarea textelor (TEXTCAPS). Acești indicatori sunt relevanți mai ales pentru dispozitivele plotter, care s-ar putea să nu suporte unele operații de desenare.

Există și o serie de indicatori folosiți pentru a determina caracteristicile de culoare și de desenare, așa cum o arată tabelul 15.4.

TABELUL 15.4 Indicatorii *GetDeviceCaps()* transmiși pentru determinarea caracteristicilor de culoare și de desenare

Indicatorul transmis	Descrierea valorii întoarse
NUMCOLORS	Numărul de culori distincte suportate de dispozitiv
PLANES	Numărul de plane de culoare suportate de dispozitiv
BITSPIXEL	Numărul de biți folosiți pentru reprezentarea unui pixel
NUMPENS	Numărul de penițe suportate de dispozitiv
NUMBRUSHES	Numărul de pensule suportate de dispozitiv
NUMFONTS	Numărul de fonturi suportate de dispozitiv
COLORRES	Rezoluția de culoare a dispozitivului sub formă de biți pe pixel, dacă se aplică
SIZEPALETTE	Numărul de intrări din paleta sistem, dacă se aplică

Parametrii funcției *GetDeviceCaps()*

Pe lângă indicatorii prezentați aici, există mulți alții care pot fi utilizați. Majoritatea acestora au rolul de a determina dacă un dispozitiv suportă anumite operații de desenare. Pentru dispozitivele de afișare și tipărire mai uzuale, operațiile de desenare sunt suportate și funcționează fără probleme.

Să folosim *GetDeviceCaps()* pentru a determina atributele de afișare pe ecran. Caseta de dialog *About* a proiectului *MapMode* poate fi un loc nimerit pentru afișarea acestor informații sub forma unei casete cu listă.

Adăugarea unui control casetă cu listă în caseta de dialog *About*

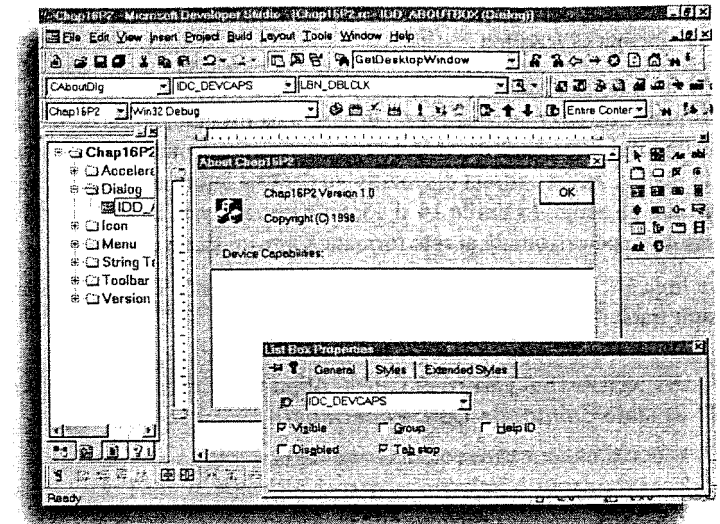
1. Selectați pagina *ResourceView* a secțiunii spațiului de lucru al proiectului.
2. Deschideți cataloagele corespunzătoare pentru a afișa caseta de dialog *IDD_ABOUTBOX*.
3. Efectuați un dublu clic pe *IDD_ABOUTBOX* pentru a edita caseta de dialog.
4. Sporiți dimensiunea casetei de dialog și adăugați o casetă cu listă.
5. Apăsați **Alt+Enter** pentru a accesa proprietățile casetei cu listă.
6. Introduceți *IDC_DEVCAPS* în câmpul **ID** din pagina **General**.
7. Efectuați un clic pe pagina **Styles** și deselectați opțiunea **Sort**.
8. Apăsați **Enter** pentru a închide caseta de dialog cu proprietăți.
9. Adăugați un control etichetă statică deasupra casetei cu listă și introduceți textul *Device Capabilities*.

Caseta de dialog *About* cu caseta de listă ale cărei proprietăți au fost stabilite ar trebui să arate ca în figura 15.4.

Acum trebuie să mapați pe caseta cu listă o variabilă control corespunzătoare, astfel încât să puteți adăuga elemente în cadrul său.

FIGURA 15.4

Editarea casetei de dialog *About* din proiectul *MapMode*.



Maparea unei variabile pe controlul casetă cu listă prin intermediul *ClassWizard*

1. Efectuați un clic pe caseta cu listă pentru a o selecta în cadrul editorului de resurse.
2. Apăsați **Ctrl+W** pentru a apela *ClassWizard*. În lista *Control IDs* din pagina **Member Variables** ar trebui să apară selectat *IDC_DEVCAPS*.
3. Efectuați un clic pe butonul **Add Variable** pentru a afișa caseta de dialog *Add Member Variable*.
4. Derulați caseta combinată **Category** și înlocuiți elementul selectat **Value** cu **Control**.
5. Introduceți *m_listDevCaps* în caseta **Member Variable**.
6. Efectuați un clic pe **OK** pentru a închide caseta de dialog.
7. Efectuați un clic pe **OK** pentru a închide *ClassWizard*.

Acum aveți în clasa *CAboutDlg* o variabilă de tip *CListBox* mapată peste controlul casetă cu listă. Este momentul să introduceți codul care va efectua interogarea unui context dispozitiv și va afișa rezultatele în caseta cu listă. În acest scop trebuie să adăugați în clasa *CAboutDlg* o funcție de tratare *OnInitDialog()* care să inițializeze caseta cu listă la deschiderea dialogului.

Adăugarea funcției de tratare *OnInitDialog()*

1. Selectați pagina *ClassView* a secțiunii spațiului de lucru al proiectului.
2. Efectuați un clic cu butonul drept pe clasa *CAboutDlg* pentru a apela meniul de context.
3. Selectați opțiunea **Add Windows Message Handler**.
4. Selectați mesajul *WM_INITDIALOG* din lista **New Windows Messages/Events**.

5. Efectuați un clic pe butonul **Add and Edit** pentru a adăuga funcția de tratare și a începe să o editați.

Acum puteți să adăugați codul prezentat în listingul 15.9, cod care înscrie rezultatele furnizate de `GetDeviceCaps()` în caseta cu listă din caseta de dialog **About**. În linia 8 a listingului se creează un obiect `CClientDC` care capturează contextul dispozitiv al casetei de dialog. Acest context dispozitiv va conține informații referitoare la dispozitivul de afișare pe ecran. În liniile 14 și 15, valoarea asociată cu indicatorul `HORZRES` reprezintă rezoluția pe orizontală și este formatată într-un șir de caractere.

În linia 16, șirul creat este adăugat în caseta cu listă pentru a fi afișat. Prin același procedeu sunt tratate toate caracteristicile vizate.

LISTINGUL 15.9 LST16_9.CPP – Afișarea în caseta de dialog **About** a rezultatelor furnizate de funcția `GetDeviceCaps()`

```
1 BOOL CAboutDlg::OnInitDialog()
2 {
3     CDialog::OnInitDialog();
4
5     // TODO: Add extra initialization here
6
7     // ** Capturam contextul dispozitiv al dialogului About
8     CClientDC dcDev(this);
9
10    // ** Declaram un sir de caractere
11    CString strCap;
12
13    // ** -Determinam si formatam caracteristicile dispozitivului
14    strCap.Format("Horizontal Resolution = %d pixels",
15                dcDev.GetDeviceCaps(HORZRES)); — ❶
16    m_listDevCaps.AddString(strCap);
17
18    strCap.Format("Vertical Resolution = %d pixels",
19                dcDev.GetDeviceCaps(VERTRES));
20    m_listDevCaps.AddString(strCap);
21
22    strCap.Format("Horizontal Size = %d cm",
23                dcDev.GetDeviceCaps(HORZSIZE) / 10);
24    m_listDevCaps.AddString(strCap);
25
26    strCap.Format("Vertical Size = %d cm",
27                dcDev.GetDeviceCaps(VERTSIZE) / 10);
28    m_listDevCaps.AddString(strCap);
29
```

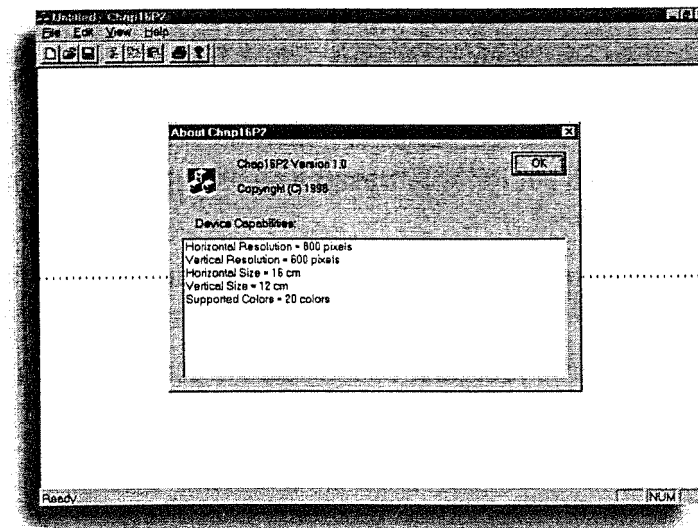
```
30    strCap.Format("Supported Colors = %d colors",
31                dcDev.GetDeviceCaps(NUMCOLORS));
32    m_listDevCaps.AddString(strCap);
33
34    return TRUE; // return TRUE unless you set
                 // the focus to a control
35    // EXCEPTION: OCX Property Pages should?
                 // return FALSE
36 }
```

- ❶ `GetDeviceCaps()` este apelată pentru a determina diferite caracteristici ale dispozitivului legat de contextul dispozitiv. Informațiile sunt adăugate într-o casetă cu listă.

Dacă asamblați și rulați aplicația după efectuarea acestor modificări și apoi efectuați un clic pe meniul **Help** și selectați opțiunea **About Map Mode...**, ar trebui să vedeți caracteristicile dispozitivului de afișare pe ecran, așa cum se ilustrează în figura 15.5. Nu vă faceți probleme dacă aveți impresia că dispozitivul de afișare permite mai multe culori decât este afișat în caseta cu listă. Totul se întâmplă deoarece contextul dispozitiv vă furnizează numărul de culori din *paleta sistem*, care este implicit 20. Acest număr poate fi mărit prin selectarea în contextul dispozitiv a unei palete mai mari, putând ajunge până la numărul maxim de culori suportat.

FIGURA 15.5

Afișarea caracteristicilor dispozitivului de afișare, furnizate de `GetDeviceCaps()`.



VEDEȚI...

- Pentru a determina caracteristicile unui dispozitiv de imprimare, vedeți pagina 512.

Utilizarea penițelor și a pensulelor

Utilizarea penițelor la trasarea de diferite linii și figuri cu diverse stiluri

Utilizarea pensulelor pentru umplerea figurilor cu diferite culori și texturi

Exploatarea funcțiilor de desenare în scopul trasării ușoare a unor figuri complexe

Crearea penițelor

Penițele reprezintă unul dintre obiectele GDI (Graphics Device Interface – interfață cu dispozitivele grafice) de bază; ele sunt implementate chiar în inima sistemului Windows și, practic, se află acolo de la bun început. Înainte de a desena ceva, trebuie să creați sau să selectați o peniță potrivită pentru operația vizată.

Penițele și figurile umplute

Penițele sunt folosite pentru trasarea de linii și curbe. Atunci când desenați figuri umplute, penițele sunt utilizate la trasarea conturului, în timp ce pensulele sunt responsabile de umplerea interiorului. Pot exista situații în care nu doriți să umpleți interiorul figurii, ci doar să trasați conturul. Acest efect poate fi obținut prin utilizarea unei pensule transparente sau vide împreună cu funcțiile de desenare a figurilor umplute. Despre acest procedeu vom vorbi mai târziu în capitolul de față.

Utilizarea clasei *CPen*

Din fericire, MFC oferă o clasă de încapsulare, numită *CPen*, care simplifică manipularea penițelor (ca obiecte de desenare). Această clasă conține obiectul GDI aflat la bază și se ocupă de alocarea și eliberarea acestuia.

Pentru crearea unei penițe trebuie să declarați un obiect corespunzător și să transmiteți niște parametri de inițializare. Linia de cod următoare creează o peniță continuă de culoare roșie:

```
CPen penRed(PS_SOLID, 3, RGB(255, 0, 0));
```

Stabilirea tipului de peniță

Primul parametru, *PS_SOLID*, reprezintă tipul peniței, fiind posibilă crearea unei varietăți de penițe prin intermediul tipurilor disponibile. Tabelul 16.1 prezintă câteva dintre stilurile existente.

TABELUL 16.1 Valori pentru tipuri de penițe

Tipul peniței	Tipul liniilor desenate
<i>PS_SOLID</i>	Linii continue
<i>PS_DASH</i>	Linii întrerupte (segmentate)
<i>PS_DOT</i>	Linii întrerupte (punctate)
<i>PS_DASHDOT</i>	Linii întrerupte (de tip linie-punct)

Modificarea grosimii peniței

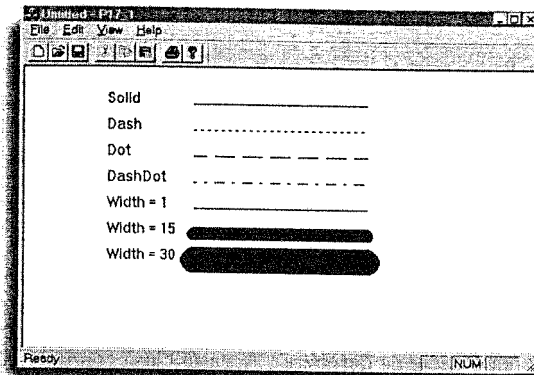
Grosimea sau lățimea peniței poate fi modificată prin intermediul celui de-al doilea parametru. Valoarea 1 pentru acest parametru va avea ca rezultat o peniță a cărei lățime este de un singur pixel – cea mai subțire posibilă. În cazul în care fixați acest parametru la 30 veți obține o peniță groasă, îndesată.

Limitări privind stilurile de linie întreruptă

Rețineți că stilurile de linie întreruptă sunt valabile numai pentru penițe de grosime unitară; orice altă grosime mai mare are ca efect o linie continuă. Figura 16.1 înfățișează diferite stiluri și grosimi de peniță în acțiune.

FIGURA 16.1

Grosimi și stiluri de penițe.

**Modificarea culorii peniței**

Al treilea parametru care poate fi specificat la crearea peniței este culoarea. Valoarea necesară aici este de tipul `COLORREF`. În formă brută, o valoare `COLORREF` nu este altceva decât un număr pe 32 de biți care combină componentele de roșu, verde și albastru ce formează o culoare. Specificarea directă a valorilor `COLORREF` este mai delicată; de exemplu, valoarea pentru un mov deschis este 16711935. Din fericire, există o macrodefiniție numită `RGB` care vă vine în ajutor. Iată cum se utilizează în cod macrodefiniția `RGB`:

```
COLORREF rgbPurple = RGB(255,0,255);
CPen penDashDotPurple(PS_DASHDOT,1,rgbPurple);
```

Limitări hardware privind culorile

Deși codificarea culorilor pe 24 de biți (True Color) vă permite să operați cu 16,7 milioane de culori și nuanțe, culorile afișate efectiv depind de caracteristicile plăcii grafice și ale monitorului. În cazul în care modul grafic curent suportă mai puține culori (de exemplu, 16, 256 sau 65536), Windows va încerca să compenseze prin crearea unei palete de culori care să reprezinte cât mai bine culorile afișate curent de sistem. Uneori va recurge la amestecarea a două culori într-un model hașurat de pixeli cu scopul de a produce pe baza culorilor disponibile un rezultat cât mai apropiat de culoarea dorită.

Macrodefiniția `RGB` primește trei parametri care reprezintă intensitățile componentelor de roșu, verde și albastru pentru culoarea dorită. Fiecare dintre acești parametri poate lua valori între 0 și 255, unde 0 înseamnă lipsa culorii și 255 înseamnă intensitatea maximă a acesteia. Prin intermediul acestor valori puteți să specificați un număr uriaș de culori și

nuanțe (16,7 milioane) – cu condiția ca placa grafică să le suporte. Listingul 16.1 conține câteva exemple de penițe colorate și valori RGB.

LISTINGUL 16.1 LST17_1.CPP – Penițe cu diferite culori și stiluri

```
1 // Penița continuă de culoare verde închis
2 CPen penSolidDullGreen(PS_SOLID,1,RGB(0,128,0)); — 1
3
4 // Penița groasă de culoare galbenă
5 CPen penSolidYellow30(PS_SOLID,30,RGB(255,255,0)); — 2
6
7 // Penița punctată de culoare roșie
8 CPen penDotRed(PS_DOT,1,RGB(255,0,0)); — 3
9
10 // Penița segmentată de culoare albastră
11 CPen penDashBlue(PS_DASH,1,RGB(0,0,255)); — 4
12
```

1 Este creată o peniță continuă, de culoare verde și de grosime 1.

2 Este creată o peniță continuă, de culoare galbenă și de grosime 30.

3 Este creată o peniță punctată, de culoare roșie și de grosime 1.

4 Este creată o peniță segmentată, de culoare albastră și de grosime 1.

Utilizarea penițelor din stoc

Nu este întotdeauna necesar să specificați caracteristicile unei penițe, deoarece Windows are disponibile câteva *penițe de stoc*. Acestea sunt configurate deja cu parametri implicați și sunt utilizate frecvent la desenarea suprafeței de lucru și a controalelor. Pentru a putea utiliza una dintre aceste penițe din stoc veți declara un obiect `CPen`, dar nu veți mai transmite nici un parametru. Veți apela apoi `CreateStockObject()`, care va stabili caracteristicile peniței la cele ale obiectului din stoc specificat. Liniile următoare apelează la `CreateStockObject()` pentru a alege din stoc o peniță neagră:

```
CPen stockBlackPen;
stockBlackPen.CreateStockObject(BLACK_PEN);
```

Utilizarea peniței nule

Stilul `NULL_PEN` ar putea părea la început puțin ciudat, pentru că liniile și curbele trasate nu vor fi vizibile; fundalul nu este modificat. Și totuși, veți avea nevoie de o peniță nulă pentru a desena figuri umplute (precum elipse sau dreptunghiuri) care să nu aibă contur.

Sunt disponibile mai multe penițe predefinite, așa cum o arată tabelul 16.2.

TABELUL 16.2 Tipuri de penițe din stoc

Numele peniței	Efectul la desenare
BLACK_PEN	Trasează linii negre
WHITE_PEN	Trasează linii albe
NULL_PEN	Trasează cu culoarea curentă de fundal

Selectarea penițelor în cadrul contextelor dispozitiv

Înainte de a putea desena cu penițele pe care le-ați creat, trebuie să selectați penițele respective în cadrul unui context dispozitiv. Așa cum am discutat în capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”, metoda `OnDraw` a unei clase reprezentare pune la dispoziție mecanismul necesar.

Un context dispozitiv are un slot pentru penițe în care se poate afla o singură peniță la un moment dat. Aceasta este penița cu care desenați. La selectarea unei noi penițe în contextul dispozitiv, penița anterioară este eliberată. Pentru instalarea noii penițe puteți apela la metoda `SelectObject()`. Așa cum am spus, penița veche se pierde și va trebui să o prindeți. Listingul 16.2 arată cum se procedează de obicei.

LISTINGUL 16.2 LST17_2.CPP – Selectarea penițelor

```

1 void MyView::OnDraw(CDC* pDC)
2 {
3     CPen penSolid(PS_SOLID,5,RGB(0,200,0)); // Penita verde
4     CPen *pOldPen = NULL;
5
6     pOldPen = pDC->SelectObject(&penSolid); — ❶
7
8     // . . . Desenam cu penita verde
9
10    pDC->SelectObject(pOldPen); — ❷
11 }
12
```

❶ În linia 6 selectăm noua peniță verde și reținem penița originală din contextul dispozitiv.

❷ Linia 10 selectează la loc vechea peniță a contextului dispozitiv.

Funcția `SelectObject()` primește un pointer la noua peniță (observați folosirea operatorului `&` în linia 6 a listingului 16.2) și întoarce un pointer la penița selectată anterior. Apoi puteți să desenați în cadrul contextului dispozitiv. După ce ați terminat, trebuie să puneți la loc penița veche; altfel, s-ar putea ca Windows să înceapă să deseneze butoane și controale cu penița dumneavoastră!

Dacă aveți două penițe, trebuie de asemenea să folosiți `SelectObject()` pentru a le schimba între ele atunci când vreți să desenați cu fiecare, așa cum o ilustrează listingul 16.3.

LISTINGUL 16.3 LST17_3.CPP – Selectarea alternativă a două penițe

```

1 void MyView::OnDraw(CDC* pDC)
2 {
3     CPen penGreen(PS_SOLID,5,RGB(0,255,0)); // Penita verde
4     CPen penBlue(PS_SOLID,3,RGB(0,0,255)); // Penita albastra
5     CPen *pOldPen = NULL;
6
7     pOldPen = pDC->SelectObject(&penGreen); // Schimbam penita
8
9     // ... Desenam cu penita verde
10
11    pDC->SelectObject(&penBlue); // Schimbam penita — ❶
12    // Functia intoarce penita verde.
13
14    // ... Desenam cu penita albastra
15    pDC->SelectObject(pOldPen); // Refacem penita originala
16 }
```

❶ Linia 11 înlocuiește penița verde selectată curent (din linia 7) cu noua peniță albastră.

Atunci când vreți să desenați cu mai multe penițe, ca în listingul 16.3, nu trebuie să rețineți decât penița originală. Nu este nevoie să rețineți pointerii întorși pe măsură ce selectați propriile penițe, deoarece oricum dumneavoastră i-ați creat și, prin urmare, îi aveți la dispoziție. La sfârșit va trebui să selectați la loc penița originală pentru a vă asigura că propriile penițe au fost eliberate din contextul dispozitiv.

Ștergerea penițelor

Odată ce nu mai aveți nevoie de penițele pe care le-ați creat, trebuie să le ștergeți. În acest fel este eliberat obiectul GDI aflat dedesubt și sunt disponibilizate resurse prețioase ale sistemului. Clasa `CPen` se ocupă automat de toate acestea la distrugerea sa. Dar ați putea dori să efectuați explicit aceste operații în cazul în care vreți să refolosiți un obiect `CPen` cu alți parametri (apelând `CreatePen()`).

Indicatorii către obiectele grafice

Deși MFC oferă clase elegante care încapsulează obiectele grafice, așa cum este și `CPen`, obiectele propriu-zise sunt identificate prin indicatori (valori întregi pe 32 de biți) care sunt variabile membru ale claselor MFC de încapsulare. De fiecare dată când apelați o funcție `SelectObject()` sau `DeleteObject()` a unui obiect context dispozitiv, un astfel de indicator către un obiect grafic este de fapt selectat sau șters la nivelul nucleului Win32.

Alternativ, dacă ați creat multe penițe și știți că unele nu mai sunt necesare, ați putea dori să eliberați resurse ale sistemului înainte ca penițele să fie șterse la sfârșitul funcției.

Funcția membru care distruge obiectul GDI aflat la bază este `DeleteObject()`. Listingul 16.4 prezintă un obiect peniță care este creat, folosit, șters și creat din nou.

LISTINGUL 16.4 LST_17_4.CPP – Utilizarea funcțiilor *DeleteObject()* și *CreatePen()*

```

1 void MyView::OnDraw(CDC* pDC)
2 {
3     CPen penMainDraw(PS_DASH,1,RGB(128,255,255));
4     CPen *pOldPen = NULL;
5
6     pOldPen = pDC->SelectObject(&penMainDraw);
7
8     // Desenam cu penita segmentata...
9
10    pDC->SelectObject(pOldPen);
11
12    penMainDraw.DeleteObject(); — ❶
13    penMainDraw.CreatePen(PS_SOLID,5,RGB(200,20,5));
14
15    pOldPen = pDC->SelectObject(&penMainDraw);
16
17    // Desenam cu penita continua
18
19    pDC->SelectObject(pOldPen);
20 }
21 // DeleteObject este apelata automat atunci cand
22 // penMainDraw este sters odata cu incheierea functiei

```

❶ Linia 12 șterge obiectul GDI aflat la bază, iar obiectul CPen poate fi refolosit ulterior pentru crearea unei alte penițe, așa cum se vede în linia 13.

În listingul 16.4, penița penMainDraw este creată la începutul funcției OnDraw(), în linia 3, după care este selectată și se efectuează operații de desenare obișnuite. Nu apălați niciodată DeleteObject() pentru o peniță care este încă utilizată de un context dispozitiv. În listing, penița originală este selectată la loc în contextul dispozitiv, permițând astfel apelul DeleteObject() pentru penița „segmentată”.

Aveți grijă când apălați DeleteObject()

Dacă apălați funcția DeleteObject() pentru o peniță în timp ce aceasta este încă selectată într-un context dispozitiv, obiectul GDI corespunzător va fi distrus și orice operații de desenare ulterioare care implică respectiva peniță pot duce la prăbușirea aplicației sau la rezultate neașteptate.

Prin ștergere, obiectul GDI corespunzător este eliberat și puteți să refolosiți clasa CPen pentru a crea o altă peniță. În continuare se apelează CreatePen() pentru crearea unei penițe noi, care este continuă. Această funcție membru are aceiași parametri cu funcția constructor: CreatePen(nPenStyle, nWidth, crColor), fiind necesară specificarea stilului, a grosimii și a culorii.

Trasarea de linii și figuri cu ajutorul penițelor

Pentru a desena ceva cu ajutorul unei penițe (sau al oricărui alt obiect GDI) aveți nevoie de un context dispozitiv în care să desenați. Așa cum ați văzut în capitolul 15, contextul dispozitiv oferă mecanismul pentru desenarea cu aceleași obiecte în cadrul unei întregi mulțimi de dispozitive, rezultatele obținute fiind identice.

Crearea unui context dispozitiv pentru desenare

Fereastra reprezentare dintr-o aplicație SDI poate fi un suport ideal pentru tendințele dumneavoastră artistice. În cele ce urmează veți vedea cum poate fi creată o aplicație SDI care folosește o reprezentare simplă destinată desenării (prin intermediul clasei de bază CView).

Alegerea contextului dispozitiv pentru desenare

Funcțiile de desenare pot fi aplicate în orice context dispozitiv valid. Aceasta înseamnă că ați putea să accesați fereastra unei alte aplicații, să apălați GetDC() pentru a-i determina contextul dispozitiv și să începeți să desenați în cadrul acestuia. Mai mult, ați putea să acaparați contextul dispozitiv pentru întreaga suprafață de lucru din Windows și să desenați aici. Programele civilizate, însă, deschid o fereastră proprie și mărginesc operațiile de desenare la suprafața acesteia. Windows va decupa rezultatele funcțiilor de desenare astfel încât acestea să nu altereze alte ferestre.

Crearea unui context dispozitiv în cadrul unei aplicații SDI

1. Efectuați un clic pe meniul **File** și selectați **New**.
2. Selectați pagina **Projects**. Din lista cu tipurile de proiecte alegeți **MFC AppWizard (exe)**.
3. Acum efectuați un clic în caseta **Project Name** și introduceți numele proiectului (în listingurile prezentate se folosește numele MyDraw).
4. Efectuați un clic pe **OK**. Va fi afișată caseta de dialog MFC AppWizard – Step 1.
5. Selectați **Single Document** și efectuați un clic pe **Finish**. În consecință, va fi creată infrastructura unei aplicații SDI care va utiliza clasa de bază CView pentru a deriva clasa reprezentare a noii aplicații.
6. Efectuați un clic pe **OK** în caseta de dialog New Project Information și AppWizard va crea noul proiect și fișierele sale sursă.

Acum sunteți gata să desenați.

Deplasarea poziției peniței

Un context dispozitiv păstrează coordonatele unei poziții curente a peniței. La desenarea liniilor, acestea sunt trasate între poziția curentă și poziția specificată, după care poziția specificată devine noua poziție curentă.

Contextele dispozitiv dispun de o funcție membru, MoveTo(), care stabilește această poziție curentă. MoveTo() se apelează cu doi parametri care specifică poziția pe orizontală și pe verticală.

Determinarea poziției de desenare curente

Poziția de desenare curentă poate fi determinată prin apelul funcției `GetCurrentPosition()` a contextului dispozitiv, aceasta întorcând poziția curentă sub forma unui obiect `CPoint`.

Selectați pagina `ClassView` a spațiului de lucru al proiectului. Ar trebui să vedeți clasa `CMyDrawView`. Efectuați un clic pe semnul plus alăturat pentru a afișa membrii acestei clase. Pentru implementarea codului de desenare puteți folosi funcția membru `OnDraw()`. Efectuați un dublu clic pe `OnDraw()` și veți vedea implementarea ilustrată în listingul 16.5.

LISTINGUL 16.5 LST17_5.CPP – Funcția `OnDraw()` standard

```
1 // CMyDrawView drawing
2
3 void CMyDrawView::OnDraw(CDC* pDC)
4 {
5     CMyDrawDoc* pDoc = GetDocument();
6     ASSERT_VALID(pDoc);
7
8     // TODO: add draw code for native data here
9 }
```

Precum vedeți, funcția `OnDraw()` primește un pointer la un context dispozitiv, acesta fiind asociat zonei client a ferestrei reprezentare a aplicației. În linia 5, funcția determină un pointer la clasa document; motivul este faptul că majoritatea aplicațiilor desenează pe baza datelor aflate în cadrul documentului. Aici nu va desena nimic. Dacă efectuați un clic pe meniul **Build** și selectați **Execute MyDraw.exe**, proiectul va fi asamblat și la rulare va afișa o fereastră goală.

Adăugați codul pentru desenare, începând cu a deplasa poziția cursorului grafic. În acest scop, adăugați după comentariul `//TODO: add draw code` apelul `MoveTo()` înfățișat în linia 8 a listingului 16.6.

LISTINGUL 16.6 LST17_6.CPP – Utilizarea funcției `MoveTo()`

```
1 void CMyDrawView::OnDraw(CDC* pDC)
2 {
3     CMyDrawDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     pDC->MoveTo(50,100); // ** Noua linie cu apelul MoveTo ** — ❶
9 }
```

❶ Linia 8 modifică poziția curentă a cursorului grafic din cadrul contextului dispozitiv prin intermediul funcției `MoveTo()`.

Deși nu se desenează nimic, poziția cursorului grafic din cadrul contextului dispozitiv se află acum la 50 de pixeli în dreapta și 100 de pixeli mai jos față de colțul stânga sus al reprezentării, aceasta datorită apelului `MoveTo()` din linia 8.

Desenarea de linii

Contextele dispozitiv oferă o funcție `LineTo()`, asemănătoare funcției membru `MoveTo()`. Parametrii primiți sunt aceiași și efectul este trasarea unei linii de la poziția curentă la poziția specificată.

Dacă adăugați în funcția `OnDraw()` codul prezentat în listingul 16.7, aplicația va desena un triunghi punctat cu ajutorul funcției `LineTo()`.

LISTINGUL 16.7 LST17_7.CPP – Desenarea de linii

```
1 void CMyDrawView::OnDraw(CDC* pDC)
2 {
3     CMyDrawDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here.
7
8     // Cream o penita
9     CPen penRed(PS_DOT,1,RGB(255,0,0));
10    // Declaram un pointer care va retine penita originala
11    CPen *pOldPen = NULL;
12
13    // Selectam penita rosie
14    pOldPen = pDC->SelectObject(&penRed);
15
16    pDC->MoveTo(50,100);
17
18    // Desenam latura 'de baza'
19    pDC->LineTo(100,100); — ❶
20    // Desenam prima latura 'laterala'
21    pDC->LineTo(75,50);
22    // Desenam a doua latura 'laterala'
23    pDC->LineTo(50,100);
24
25    // Selectam la loc penita originala
26    pDC->SelectObject(pOldPen);
27 }
```

❶ Liniiile de la 19 la 23 trasează cele trei linii care formează triunghiul cu ajutorul funcției `LineTo()`.

Ca mai devreme, trebuie să creați o peniță (roșie și punctată) și apoi să o selectați în contextul dispozitiv. După aceea se deplasează cursorul grafic în poziția (50,100), prin apelul `MoveTo()` din linia 16. Funcția `LineTo()` care urmează (linia 19) va trasa o linie

între punctele (50,100) și (100,100). Segmentul va fi orizontal, între coordonatele 50 și 100; coordonata pe verticală rămâne aceeași (50). Următoarele două apeluri `LineTo()` (din liniile 21 și 23) trasează de la (100,100) la (75,50) și, mai departe, la (50,100).

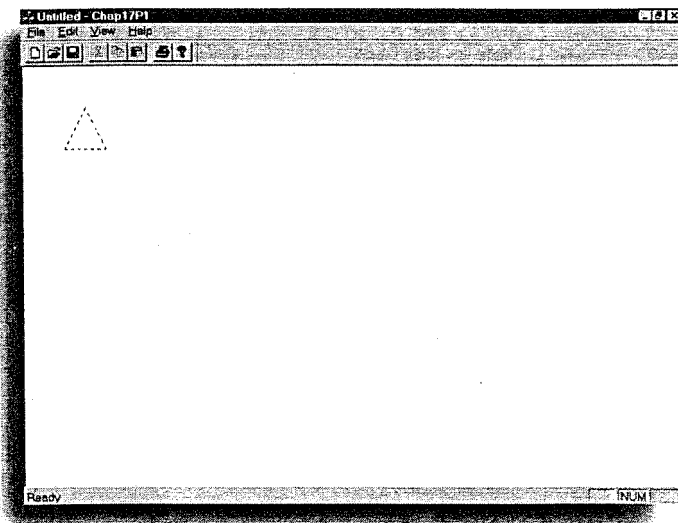
Pentru a asambla și rula aplicația astfel modificată, efectuați un clic pe meniul **Build** și selectați **Execute**, după care efectuați un clic pe **Yes** pentru a confirma necesitatea de asamblare a proiectului. Dacă totul merge bine, ar trebui să obțineți un mic triunghi punctat care seamănă cu cel înfățișat în figura 16.2.

Desenarea prin puncte de coordonate

În capitolul 8, „Tratarea evenimentelor generate de mouse”, v-ați întâlnit cu clasa punct de coordonate, pe care ați folosit-o pentru a reține și transmite perechi de coordonate. Utilizarea clasei `CPoint` este o modalitate facilă de a reține și manipula împreună coordonatele pe orizontală și verticală. Funcțiile `MoveTo()` și `LineTo()` ale contextelor dispozitiv pot primi ca parametri obiecte `CPoint`, așa că puteți folosi la desenare astfel de obiecte ca alternativă la transmiterea distinctă a coordonatelor pe cele două axe. O funcție utilă este `GetCurrentPosition()`, care întoarce un obiect `CPoint` ce conține poziția curentă a cursorului grafic (acolo unde a ajuns ultimul apel `MoveTo()` sau `LineTo()`).

FIGURA 16.2

Un triunghi desenat cu ajutorul funcției `LineTo()`.



Există, de asemenea, o clasă `CRect` care conține două obiecte `CPoint`; unul dintre acestea corespunde colțului stânga sus al unui dreptunghi, iar celălalt reprezintă colțul dreapta jos. Puteți apela `GetClientRect()` pentru a determina dreptunghiul care încadrează fereastra reprezentării, reținând rezultatul într-un obiect `CRect`. `CRect` oferă, de altfel, o serie de funcții membru utile pentru manipularea informațiilor despre dreptunghiul corespunzător. Una dintre aceste funcții membru este `CenterPoint()`, care întoarce un obiect `CPoint` ce

conține coordonatele punctului din centrul dreptunghiului. În cazul zonei client, acest punct reprezintă chiar centrul ferestrei reprezentare SDI.

Determinarea dreptunghiului care încadrează întreaga fereastră

Dacă doriți să obțineți coordonatele pentru întreaga fereastră, puteți să apelați funcția `GetWindowRect()`, transmitând un pointer la un obiect `CRect`. Coordonatele ecran ale ferestrei vor fi înscrise în obiectul `CRect` transmis. Dar dacă desenați în cadrul contextului dispozitiv primit de funcția `OnDraw()`, rezultatele vor fi în continuare decupate la zona client.

`CRect` poate furniza, de asemenea, lățimea și înălțimea dreptunghiului prin intermediul funcțiilor membru `Width()` și `Height()`.

Puteți să înlăturați codul care desenează triunghiul și să modificați funcția `OnDraw()` în concordanță cu listingul 16.8, rezultatul fiind modelul înfățișat în figura 16.3.

LISTINGUL 16.8 LST17_8.CPP – Desenarea cu ajutorul claselor `CPoint` și `CRect`

```
1 void CMyDrawView::OnDraw(CDC* pDC)
2 {
3     CMyDrawDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // Cream o penita
9     CPen penRed(PS_DOT, 1, RGB(255, 0, 0));
10    CPen *pOldPen = NULL;
11
12    pOldPen = pDC->SelectObject(&penRed);
13
14    // Determinam coordonatele zonei de desenare
15    CRect rcClient;
16    GetClientRect(&rcClient);
17
18    for (int x=0; x < rcClient.Width(); x+=5)
19        for (int y=0; y < rcClient.Height(); y+=5)
20        {
21            CPoint ptNew(x,y);
22            pDC->MoveTo(rcClient.CenterPoint());
23            pDC->LineTo(ptNew);
24        }
25
26    pDC->SelectObject(pOldPen);
27 }
```

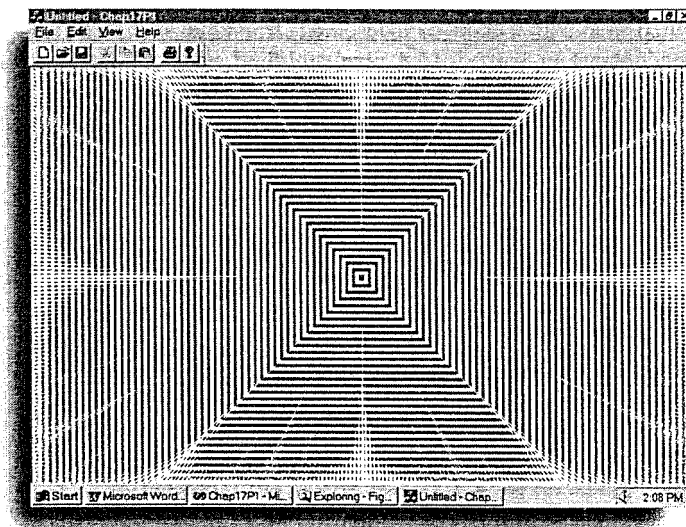
① Linii de la 18 la 24 parcurg zona client pe orizontală și pe verticală, din cinci în cinci pixeli, trasând linii din centrul ferestrei către fiecare din punctele parcurse.

Observați cum liniile 15 și 16 solicită clasei reprezentare să înscrie în variabila `rcClient` informații privind zona client, fiind apelată funcția `GetClientRect()`. În acest fel devine ușoară parcurgerea pe orizontală și pe verticală a dreptunghiului cu ajutorul a două bucle `for`. Mai întâi `x` parcurge lățimea dreptunghiului din cinci în cinci pixeli. Apoi, pentru fiecare valoare a lui `x`, `y` parcurge înălțimea dreptunghiului tot cu câte cinci pixeli. Astfel se acoperă o rețea rectangulară; liniile sunt trasate de la centru (obținut prin apelul `CenterPoint()` din linia 22) către punctul curent din rețea, care este creat în variabila `ptNew` în linia 21 și transmis funcției `LineTo()` în linia 23.

După efectuarea acestor modificări puteți să asamblați și să rulați programul. Cu numai câteva linii de cod, rezultatul poate fi o operă de artă modernă (vedeți figura 16.3).

FIGURA 16.3

Desenarea de linii cu puncte de coordonate și dreptunghiuri.



Desenarea de cercuri și elipse

Un cerc nu este decât o elipsă care are înălțimea egală cu lățimea, așa că ambele figuri sunt trasate de o aceeași funcție membru a contextului dispozitiv. Funcția `Ellipse()` are două forme. Prima formă primește un dreptunghi transmis ca obiect `CRect`. Cea de-a doua formă necesită patru parametri, reprezentând coordonatele pe orizontală și pe verticală ale colțului stânga sus și ale colțului dreapta jos.

Modificați funcția `OnDraw()` ca în listingul 16.9. Aceasta va desena acum cercuri concêntrice (vedeți figura 16.4).

Prima modificare pe care o veți observa este apelul `pDC->SelectStockObject(NULL_BRUSH)` din linia 15; în lipsa acestui apel, funcția `Ellipse()` din linia 30 ar umple figurile desenate cu pensula selectată implicit. (Despre pensule vom discuta în secțiunea „Crearea pensulelor”, mai târziu în acest capitol.) Observați că această funcție determină contextul dispozitiv să nu umple elipsa, ci să traseze doar conturul.

LISTINGUL 16.9 LST17_9.CPP – Desenarea de cercuri prin intermediul funcției *Ellipse*

```
1 void CMyDrawView::OnDraw(CDC* pDC)
2 {
3     CMyDrawDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // Cream o penita
9     CPen penRed(PS_DOT,1,RGB(255,0,0));
10    CPen *pOldPen = NULL;
11
12    pOldPen = pDC->SelectObject(&penRed);
13
14    // Elipsele nu vor fi pline
15    pDC->SelectStockObject(NULL_BRUSH);
16
17    CRect rcClient;
18    GetClientRect(&rcClient);
19
20    // Cream un obiect dreptunghi centrat
21    CRect rcEllipse(rcClient.CenterPoint(),
22                   rcClient.CenterPoint());
23
24
25    // Iteram pana cand elipsele ajung sa umple fereastra — ❶
26    while(rcEllipse.Width() < rcClient.Width()
27          && rcEllipse.Height() < rcClient.Height())
28    {
29        // Marim dimensiunea elipsei cu 10 pixeli
30        rcEllipse.InflateRect(10,10);
31        pDC->Ellipse(rcEllipse);
32    }
33
34    pDC->SelectObject(pOldPen);
35 }
```

❶ Bucula `while` din linia 25 duce la execuția liniilor de la 28 la 30, desenând cercuri tot mai mari până când diametrul atinge lățimea sau înălțimea zonei client.

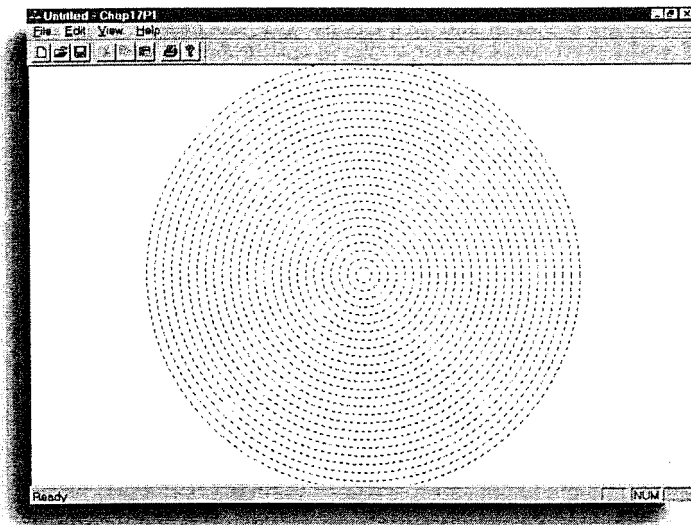
Mai departe se creează un dreptunghi (`rcEllipse`) care va fi folosit pentru desenarea elipselor, colțurile stânga-sus și dreapta-jos fiind inițializate cu centrul ferestrei client. Bucula `while` din liniile 25, 26 și 27-31 iterează până când dreptunghiul de încadrare a elipsei devine mai lat sau mai înalt decât fereastra.

În această buclă, în linia 30, este apelată o altă funcție membru a clasei `CRect`, `InflateRect()`, care crește lățimea și înălțimea dreptunghiului cu câte 10 pixeli. Efectul obținut este mărirea dreptunghiului în jurul centrului său. În interiorul dreptunghiului „în creștere” se desenează o elipsă, obținându-se astfel cercuri concentrice.

Efectul acestor modificări poate fi văzut în urma asamblării și a rulării programului, capodopera rezultată fiind înfățișată în figura 16.4.

FIGURA 16.4

Desenarea de cercuri cu `Ellipse()`.



Desenarea de curbe

`PolyBezier()` este o funcție al cărei nume vine de la matematicianul Bézier, care a redus curbele la *polinoame cubice*. Polinomul descrie linia curbă, iar faptul că este cubic înseamnă că există un punct de început, unul de sfârșit și două puncte de control înspre care este flexată curba prin interpolare. Particula *Poly* arată că este posibilă trasarea mai multor curbe conectate între ele. Funcția primește un vector de obiecte `CPoint` (cel puțin patru) și numărul de puncte din vector.

Considerente de viteză legate de trasarea curbelor

Din cauza calculului suplimentar care sunt necesare pentru trasarea liniilor curbe, timpul necesar este substanțial mai mare decât în cazul liniilor drepte. Probabil că nu veți remarca diferența de viteză dacă nu trasați decât câteva curbe, dar aceasta poate deveni semnificativă în cazul în care aplicația trebuie să deseneze mai multe curbe.

Ați putea să modificați funcția `OnDraw()` între `GetClientRect()` și apelul `SelectObject()` din final, ca în listingul 16.10, astfel încât să traseze curbele Bézier înfățișate în figura 16.5.

LISTINGUL 16.9 LST17_9.CPP – Desenarea de cercuri prin intermediul funcției `Ellipse`

```
1 GetClientRect(&rcClient);
2
3 // Cream un vector care va contine puncte
4 CPoint ptBezierAr[4];
5 CPen penBlue(PS_SOLID,3,RGB(0,0,255));
6
7 pDC->SelectObject(&penBlue); — ❶
8 for(int i=0; i < 4 ;i++)
9 {
10     // Initializam vectorul cu puncte aleatoare
11     ptBezierAr[i] =
12     CPoint(rand()%rcClient.Width(),
13           rand()%rcClient.Height());
14
15     // Afisam o eticheta — ❷
16     CString strPointLabel;
17     strPointLabel.Format("%d.",i+1);
18     pDC->TextOut(ptBezierAr[i].x,
19                ptBezierAr[i].y,
20                strPointLabel);
21
22     // Desenam punctele de coordonate cu albastru
23     CRect rcDot(ptBezierAr[i],ptBezierAr[i]);
24     rcDot.InflateRect(2,2);
25     pDC->Ellipse(rcDot);
26 }
27
28 // Trasam o curba Bezier de culoare rosie
29 pDC->SelectObject(&penRed);
30 pDC->PolyBezier(ptBezierAr,4); — ❸
31
32 pDC->SelectObject(pOldPen);
33
```

❶ În linia 7 selectăm penița albastră pentru a desena punctele cu albastru (în linia 26).

❷ Liniile de la 16 la 20 creează un obiect `CString`, în cadrul său fiind înscris numărul punctului. Acest șir este afișat apoi în poziția punctului cu scopul de a-l eticheta, prin apelul funcției `TextOut()` din linia 18.

❸ În linia 30 se trasează curba Bézier prin apelul funcției `PolyBezier()`, fiind utilizată penița roșie selectată în linia 29.

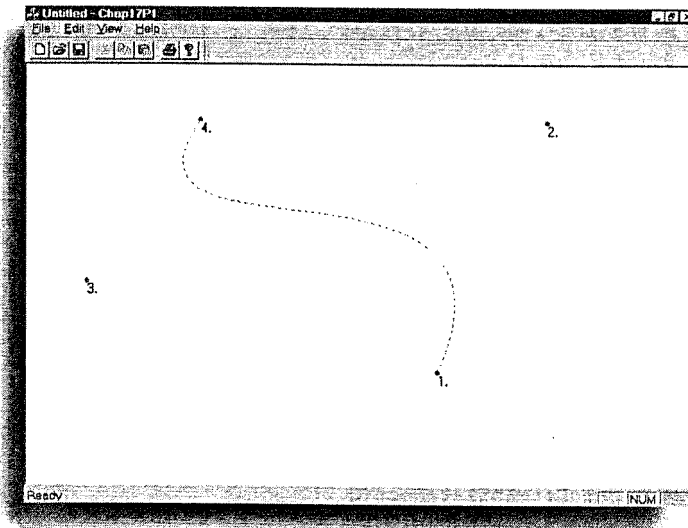
În listingul 16.10, linia 4 declară vectorul `ptBezierAr` de obiecte `CPoint` și cele patru elemente ale sale sunt inițializate la puncte aleatoare din interiorul zonei client (liniile 12 și 13). Apoi, în linia 18, funcția `TextOut()` este apelată pentru a afișa o etichetă asociată punctului curent (despre această funcție veți afla mai multe în capitolul 17, „Utilizarea fonturilor”).

Apelul funcției `Ellipse()` din linia 25 este ideal pentru desenarea unei buline albastre (cu noua peniță) care să illustreze fiecare punct. În cele din urmă, după selectarea la loc a peniței roșii, `PolyBezier()` este apelată în linia 30, transmitând vectorul de puncte și specificând că numărul acestora este 4. În consecință, va fi trasată o curbă care este mărginită de punctele 1 și 4 și este controlată de punctele 2 și 3.

Acum puteți să asamblați și să rulați programul pentru a vedea curba trasată de funcția `PolyBezier()`. Figura 16.5 ilustrează destul de clar rolul fiecărui punct în cadrul funcției. Redimensionați fereastra și observați cum de fiecare dată este trasată o altă curbă.

FIGURA 16.5

Trasarea curbelor cu
`PolyBezier()`.



Desenarea poligoanelor

`Polyline()` este identică cu `PolyBezier()` în ceea ce privește parametrii necesari. Se transmit un vector de obiecte `CPoint` și numărul de elemente din vector. În loc să traseze o curbă, această funcție unește pur și simplu punctele prin linii drepte. Puteți să specificați oricâte puncte – dar cel puțin două – și funcția se poate dovedi utilă la desenarea unor obiecte predefinite sub forma unor vectori de obiecte `CPoint`. Există, de asemenea, funcțiile `PolyLineTo()` și `PolyBezierTo()`, care stabilesc coordonatele implicite la ultimul punct desenat.

Dacă ați înlocui apelul `PolyBezier()` din funcția `OnDraw()` cu `Polyline()` și ați asambla și rula aplicația, în locul curbei Bézier veți obține segmente unite (vedeți listingul 16.11).

LISTINGUL 16.11 LST17_11.CPP – Desenarea de linii cu `Polyline()`

```
1 // Selectam penita rosie
2 pDC->SelectObject(&penRed);
3
```

```
4 // Trasam o linie poligonala cu 4 varfuri
5 pDC->Polyline(ptBezierAr,4);
6
7 // Selectam la loc penita originala
8
9 pDC->SelectObject(pOldPen);
10
```

Dacă doriți să desenați poligoane cu `Polyline()` (ca în linia 5), va trebui ca ultimul punct din vector să aibă aceleași coordonate cu primul punct.

Crearea pensulelor

Penițele sunt utile pentru trasarea conturilor de figuri, dar umplerea acestora cu culoare necesită o pensulă. Majoritatea funcțiilor de desenare GDI folosesc atât o peniță, cât și o pensulă. Penița este utilizată pentru trasarea conturului, iar pensula se folosește la desenarea interiorului figurilor. În acest fel aveți posibilitatea de a alege o peniță cu o anumită culoare și un anumit stil și, respectiv, o pensulă de altă culoare, desenând întreaga figură printr-un singur apel de funcție.

Utilizarea clasei `CBrush`

Clasa MFC care încapsulează obiectul GDI pensulă este `CBrush`. O pensulă poate fi o culoare plină, o hașură sau poate să provină dintr-un bitmap sau dintr-un șablon. Prin urmare, există constructori care primesc parametri corespunzători acestor tipuri, fiind posibil totodată să creați un obiect neinițializat și ulterior să apelați la una dintre multele funcții de creare disponibile.

Clasa `CBrush` și `HBRUSH`

Clasa MFC `CBrush` este o altă clasă de încapsulare care ascunde un indicator `HBRUSH` către un obiect GDI. Acest indicator se poate determina și utiliza în mod direct prin aplicarea unei conversii (`HBRUSH`) asupra unui obiect `CBrush`.

Crearea de pensule colorate și hașurate

Pensula care se creează cel mai ușor este pensula de culoare uniformă. Este suficient să declarați un obiect `CBrush` și să transmiteți ca unic parametru o referință de culoare pentru culoarea dorită. Astfel veți obține o pensulă care va umple suprafețele uniforme cu respectiva culoare, ca aici:

```
CBrush brYellow( RGB(192,192,0) );
```

Puteți să apelați la hașuri, construind pensula prin specificarea unui indicator de hașură ca prim parametru și a culorii dorite ca al doilea parametru, așa cum se vede mai jos:

```
CBrush brYellowHatch( HS_DIAGCROSS, RGB(192,192,0) );
```

Sunt disponibili mai mulți indicatori de hașură. Aceștia sunt prezentați în tabelul 16.3.

TABELUL 16.3 Indicatori de hașură pentru pensule

Indicator de hașură	Efect
HS_CROSS	Caroiaj orizontal și vertical
HS_DIAGCROSS	Caroiaj diagonal
HS_HORIZONTAL	Linii orizontale
HS_VERTICAL	Linii verticale
HS_BDIAGONAL	Linii oblice (din stânga-sus înspre dreapta-jos)
HS_FDIAGONAL	Linii oblice (din stânga-jos înspre dreapta-sus)

Colorarea fundalului unei ferestre

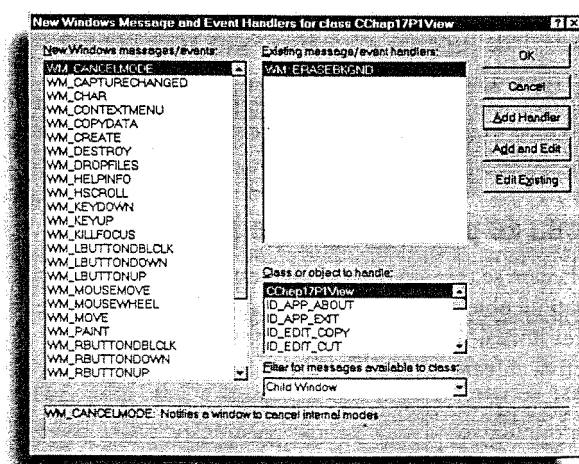
Pensulele pot fi utilizate pentru a modifica fundalul unei ferestre, prin tratarea mesajului de ștergere a fundalului (WM_ERASEBKGD) trimis de Windows.

Adăugarea unei funcții de tratare pentru modificarea culorii de fundal a unei ferestre la ștergerea fundalului acesteia

1. Efectuați un clic cu butonul drept pe bara Wizard – cea care conține trei casete combinate. Prima casetă combinată ar trebui să înfățișeze clasa reprezentare (de exemplu, CMyDrawView), iar cea de-a doua ar trebui să afișeze (All class members).
2. Va fi afișat un meniu de context care conține opțiunea Add Windows Message Handler. Selectați această opțiune. Va apărea caseta de dialog pentru tratarea mesajelor, prezentată în figura 16.6.

FIGURA 16.6

Caseta de dialog pentru tratarea mesajelor și a evenimentelor.



3. Selectați mesajul WM_ERASEBKGD și efectuați un clic pe butonul Add and Edit. În consecință, va fi creată o funcție de tratare numită OnEraseBkgnd().

4. Prin supradefinirea implementării implicite a acestei funcții de tratare puteți configura fundalul ferestrei. Listingul 16.12 prezintă un exemplu de implementare care are ca rezultat un fundal hașurat de culoare galbenă.

LISTINGUL 16.12 LST17_12.CPP – Modificarea fundalului cu OnEraseBkgnd()

```

1 BOOL CMyDrawView::OnEraseBkgnd(CDC* pDC)
2 {
3     CBrush brYellowHatch(HS_DIAGCROSS, RGB(192, 192, 0));
4
5     CRect rcClient;
6     GetClientRect(&rcClient);
7     pDC->FillRect(rcClient, &brYellowHatch); — ❶
8
9     return TRUE;
10 }
```

- ❶ În cazul în care tratați mesajul WM_ERASEBKGD, dumneavoastră vă revine sarcina de a șterge fundalul, așa cum o arată apelul FillRect() din linia 7, și de a întoarce valoarea TRUE, ca în linia 9, pentru a arăta că operația a fost îndeplinită de propria funcție.

În cazul în care compilați și rulați aplicația după efectuarea modificărilor prezentate în listingul 16.12, fundalul va apărea ca o hașură de culoare galbenă.

Prin supradefinirea funcției OnEraseBkgnd() se creează o pensulă hașurată de culoare galbenă (linia 3), se determină dreptunghiul client (linia 5) și apoi se apelează FillRect() (în linia 7) în scopul umplerii acestui dreptunghi cu pensula generată. Instrucțiunea return TRUE din linia 9 informează Windows că am rezolvat ștergerea fundalului și că nu mai este nevoie să se ocupe de acest aspect.

Crearea pensulelor pe bază de șabloane sau imagini

Posibilitatea de a crea pensule pe baza unor imagini este o facilitare destul de elegantă oferită de Windows. În acest fel puteți să acoperiți zone de ecran cu imagini dispuse alăturat. Unele aplicații profită de ocazie pentru a afișa bitmap-uri arătoase pe suprafața barelor cu instrumente.

Crearea unei imagini în scopul utilizării ca pensulă

1. Selectați pagina ResourceView din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus pentru a deschide resursele proiectului (numit, de pildă, MyView).
3. Efectuați un clic cu butonul drept pe numele proiectului și selectați opțiunea Insert din meniul contextual.
4. Selectați Bitmap din caseta de dialog Insert Resource.

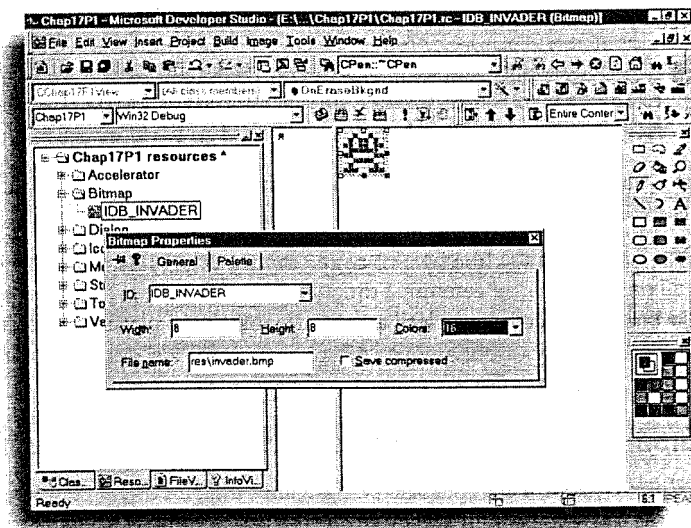
Resursele bitmap compilate

La compilarea aplicației, resursele bitmap sunt încorporate în forma finală a programului executabil (fișierul .exe). Puteți să deschideți fișiere .exe selectând Resources în caseta **Open As** pentru a vedea bitmap-urile încorporate. În cazul în care intenționați să includeți în aplicație mai multe bitmap-uri de dimensiuni mari, ar putea fi de preferat să le încărcați din fișiere (sau din bibliotecă DLL de resurse) decât să măriți dimensiunea fișierului .exe al programului.

5. Va apărea o nouă resursă de tip bitmap, numită IDB_BITMAP1. Efectuați un clic pe grila de desenare și apoi apăsați Alt+Enter pentru a afișa caseta de dialog cu proprietățile bitmap-ului, sau selectați **Properties** din meniul **View**.
6. Stabiliți un nume potrivit pentru bitmap (de pildă, IDB_INVADER pentru exemplul nostru) și fixați lățimea și înălțimea la 8 pixeli. Apoi desenați imaginea dorită pentru pensulă (figura 16.7 vă propune o schiță de extraterestru).

FIGURA 16.7

Crearea unui bitmap pentru o pensulă.



Acum trebuie să implementați codul necesar construirii unei pensule pe baza noului bitmap. În acest scop trebuie să închideți fereastra de editare a bitmap-ului și să modificați implementarea funcției `OnEraseBknd` așa cum o arată listingul 16.13.

LISTINGUL 16.13 LST17_13.CPP – Utilizarea unei pensule de tip bitmap

```
1 BOOL CMyDrawView::OnEraseBknd(CDC* pDC)
2 {
3     // Declaram un bitmap
4     CBitmap bmInvader;
5
6     // Incarcam resursa bitmap
```

```
7
8     bmInvader.LoadBitmap(IDB_INVADER); — ❶
9     // Cream o pensula pe baza bitmap-ului
10
11     CBrush brInvader(&bmInvader); — ❷
12
13     CRect rcClient;
14     GetClientRect(&rcClient);
15
16     // Umplem dreptunghiul cu pensula bitmap
17     pDC->FillRect(rcClient,&brInvader); — ❸
18
19     return TRUE;
20 }
```

❶ Bitmap-ul este încărcat prin apelul funcției `LoadBitmap()`, pe baza resursei `IDB_INVADER`.

❷ Un obiect `CBrush` poate fi construit dintr-un obiect `CBitmap` cu scopul de a utiliza bitmap-ul respectiv pe post de textură.

❸ Fundalul poate fi șters cu ajutorul funcției `FillRect()`, prin intermediul pensulei bitmap.

În cazul în care compilați și rulați programul după efectuarea modificărilor ilustrate în listingul 16.13, fundalul ferestrei ar trebui să fie compus dintr-o mulțime de extraterestri mici. Clasa `CBitmap` este folosită pentru manipularea bitmap-ului (linia 4), resursa fiind încărcată prin apelul `LoadBitmap()` (linia 8). După crearea unui obiect `CBrush` pe baza unui bitmap încărcat (linia 11), pensula poate fi utilizată apoi în operațiile de desenare obișnuite, așa cum este apelul funcției `FillRect()` din linia 17.

Limitări privind pensulele bitmap

Windows 95 nu poate folosi decât pensule de 8x8 pixeli, dar Windows NT nu are astfel de limite.

Utilizarea pensulelor din stoc

Ca și în cazul penițelor, există mai multe pensule de stoc ce pot fi selectate în orice moment. Tabelul 16.4 prezintă aceste pensule.

TABELUL 16.4 Pensule din stoc

Pensulă din stoc	Efect
BLACK_BRUSH	Pensulă neagră
WHITE_BRUSH	Pensulă albă
DKGRAY_BRUSH	Pensulă gri închis
LTGRAY_BRUSH	Pensulă gri deschis
GRAY_BRUSH	Pensulă gri
NULL_BRUSH	Transparent (nu umple figurile)

Selectarea acestor pensule într-un context dispozitiv se poate face direct, prin apelul funcției `SelectStockObject()`, așa cum a fost cazul pensulei `NULL_BRUSH` în linia 15 a listingului 16.9.

La crearea unei pensule pot fi utilizate mai multe dintre culorile definite în sistem, apelând în acest sens la funcția `CreateSysColorBrush()`. Aceste culori ale sistemului pot fi selectate prin transmiterea indicatorilor corespunzători în apelul funcției menționate. Lista acestor indicatori este prezentată în tabelul 16.5.

Utilizarea funcției `CreateSysColorBrush()`

Oricând doriți să folosiți într-o aplicație culorile definite de Windows, apelați la funcția `CreateSysColorBrush()`. În cazul în care folosiți o culoare sub forma unei valori `COLORREF` constante care corespunde culorii dorite din Windows, este posibil ca utilizatorul să modifice culorile definite în sistem, în timp ce aplicația dumneavoastră ar afișa în continuare culoarea fixată.

TABELUL 16.5 Indicatori pentru culorile sistem ce pot fi transmiși funcției `CreateSysColorBrush()`

Indicator de culoare sistem	Descriere
<code>COLOR_DESKTOP</code>	Culoarea de fond a suprafeței de lucru
<code>COLOR_BTNTEXT</code>	Culoarea etichetelor de pe butoane
<code>COLOR_GRAYTEXT</code>	Culoarea textului voalat sau dezactivat
<code>COLOR_HIGHLIGHT</code>	Culoarea de fond pentru evidențierea elementelor selectate
<code>COLOR_HIGHLIGHTTEXT</code>	Culoarea textului pentru evidențierea elementelor selectate
<code>COLOR_ACTIVEBORDER</code>	Culoarea cadrului unei ferestre active
<code>COLOR_INACTIVEBORDER</code>	Culoarea cadrului unei ferestre inactive
<code>COLOR_INFOBK</code>	Culoarea de fond a etichetelor flotante
<code>COLOR_INFOTEXT</code>	Culoarea de text a etichetelor flotante
<code>COLOR_WINDOW</code>	Culoarea de fundal a ferestrei
<code>COLOR_WINDOWFRAME</code>	Culoarea marginii unei ferestre
<code>COLOR_WINDOWTEXT</code>	Culoarea textului dintr-o fereastră
<code>COLOR_3DDKSHADOW</code>	Culoarea pentru umbra butoanelor
<code>COLOR_3DFACE</code>	Culoarea pentru suprafața butoanelor
<code>COLOR_3DHILIGHT</code>	Culoarea de evidențiere pentru butoane
<code>COLOR_3DLIGHT</code>	Culoarea pentru laturile luminate ale unui buton

Încercați să modificați funcția `OnEraseBkgnd()` pentru a crea o astfel de pensulă (vedeți listingul 16.14).

LISTINGUL 16.14 LST_17_14.CPP – Crearea unei pensule cu o culoare din sistem

```

1  BOOL CMyDrawView::OnEraseBkgnd(CDC* pDC)
2  {
3      CBrush brDesktop;
4
5      brDesktop.CreateSysColorBrush(COLOR_DESKTOP); — ❶
6
7      CRect rcClient;
8      GetClientRect(&rcClient);
9      pDC->FillRect(rcClient, &brDesktop);
10
11     return TRUE;
12 }
```

❶ Funcția membru `CreateSysColorBrush()` poate fi utilizată pentru crearea unei pensule a cărei culoare este una din culorile standard definite în sistem.

După compilarea modificărilor și rularea programului, fundalul ferestrei ar trebui să aibă culoarea curentă a fondului suprafeței de lucru.

Selectarea pensulelor în cadrul contextelor dispozitiv

Pentru selectarea unei pensule puteți apela funcția membru `SelectObject()` a contextului dispozitiv. Aceasta funcționează ca și în cazul penițelor și întoarce un pointer la pensula selectată anterior. Este bine să rețineți acest pointer întors, așa cum ați procedat la selectarea penițelor, pentru a putea restabili în final pensula originală. Toate acestea sunt ilustrate prin modificările aduse funcției `OnEraseBkgnd()` în listingul 16.15.

LISTINGUL 16.15 LST17_15.CPP – Selectarea pensulelor într-un context dispozitiv

```

1  BOOL CMyDrawView::OnEraseBkgnd(CDC* pDC)
2  {
3      CBrush brDesktop;
4      brDesktop.CreateSysColorBrush(COLOR_DESKTOP);
5
6      CRect rcClient;
7      GetClientRect(&rcClient);
8
9      CBrush* pOldBrush = pDC->SelectObject(&brDesktop);
10
11     pDC->Ellipse(rcClient); — ❶
12
13     pDC->SelectObject(pOldBrush);
14     return TRUE;
15 }
```

❶ Elipsa desenată în linia 11 este umplută cu culoarea stabilită pentru suprafața de lucru.

Din cauză că acest exemplu folosește o elipsă pentru a umple fundalul, la rularea programului se vor întâmpla unele lucruri ciudate. Pensula este selectată și se desenează o suprafață eliptică având culoarea de fundal a suprafeței de lucru, dar pentru că elipsa nu acoperă întreaga fereastră, se creează o regiune transparentă prin care se poate vedea suprafața de lucru aflată dedesubt.

După operarea acestor modificări, la compilarea și rularea programului veți vedea acest efect ciudat. Același efect de transparență ar apărea și dacă ați utiliza pensula `NULL_BRUSH` la generarea fundalului.

Ștergerea pensulelor

Asemeni penițelor, o pensulă poate fi ștersă pentru a elibera resursa asociată, fiind posibilă apoi utilizarea uneia dintre funcțiile `Create` în scopul creării unei alte pensule. Funcțiile `Create` disponibile sunt următoarele: `CreateSolidBrush()`; `CreateHatchBrush()`; `CreateBrushIndirect()`, care primește o structură `LOGBRUSH`; `CreatePatternBrush()`, care primește un bitmap; `CreateSysColorBrush()`, despre care tocmai am discutat. Toate acestea primesc parametri identici cu funcțiile constructor corespunzătoare.

Ștergerea pensulelor bazate pe bitmap-uri sau șabloane

La ștergerea unei pensule bitmap, nu uitați să ștergeți și bitmap-ul asociat (care nu este șters automat). Aveți grijă, de asemenea, ca ștergerea bitmap-ului să se producă după ștergerea pensulei, astfel încât să nu ștergeți un obiect aflat încă în uz.

Desenarea de figuri umplute cu ajutorul pensulelor

O multitudine de funcții de desenare generează figuri umplute cu ajutorul pensulelor. Pot fi desenate mai multe tipuri diferite de poligoane, sectoare de cerc sau elipsă și dreptunghiuri.

Desenarea de dreptunghiuri și dreptunghiuri rotunjite

Figurile rectangulare pot fi desenate cu ajutorul funcțiilor `Rectangle()` și `RoundRect()` (precum și `FillRect()`, pe care deja ați întâlnit-o). `Rectangle()` primește parametri identici cu `Ellipse()`; așadar, primește un obiect `CRect` sau patru întregi care specifică coordonatele colțurilor stânga-sus și dreapta-jos.

`RoundRect()` necesită încă o pereche de coordonate, primită fie ca obiect `CPoint`, fie prin doi întregi. Această pereche de coordonate precizează lățimea elipsei care este desenată în fiecare colț al dreptunghiului.

Dacă modificați `OnEraseBknd()` pentru a apela `Rectangle()` în loc de `Ellipse()`, așa cum o arată linia de cod de mai jos, atunci fundalul este desenat cu culoarea suprafeței de lucru:

```
pDC->Rectangle(rcClient);
// ** Desenam un dreptunghi cu o pensula avand culoarea suprafeței de
// lucru
```

Remarcați, totuși, că `FillRect()` avea nevoie de un pointer la o pensulă, în timp ce `Rectangle()` folosește pensula selectată curent în contextul dispozitiv.

Funcția `RoundRect()` vă permite desenarea unor dreptunghiuri cu colțurile rotunjite. Dacă modificați din nou funcția `OnDraw()` și înlocuiți codul care trasează linia poligonală în concordanță cu listingul 16.16, veți vedea nu un dreptunghi obișnuit, ci unul ale cărui colțuri sunt rotunjite.

LISTINGUL 16.16 LST17_16.CPP – Desenarea cu ajutorul funcției `RoundRect()`

```
1 void CMyDrawView::OnDraw(CDC* pDC)
2 {
3     CMyDrawDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     CPen penRed(PS_SOLID, 5, RGB(255, 0, 0));
9     CPen *pOldPen = NULL;
10
11     pOldPen = pDC->SelectObject(&penRed);
12
13     CRect rcClient;
14     GetClientRect(&rcClient);
15
16     CBrush brBlue(RGB(0, 0, 255));
17     CBrush *pOldBrush = NULL;
18
19     pOldBrush = pDC->SelectObject(&brBlue);
20
21     rcClient.DeflateRect(50, 50);
22     pDC->RoundRect(rcClient, CPoint(15, 15)); ❶
23
24     pDC->SelectObject(pOldBrush);
25     pDC->SelectObject(pOldPen);
26 }
```

❶ Linia 22 ilustrează modul în care este folosită funcția `RoundRect()` pentru a desena un dreptunghi cu o rază de rotunjire a colțurilor de 15 pixeli.

Remarcați aici că am înlocuit penița punctată de culoare roșie cu o peniță continuă, groasă, tot de culoare roșie (în linia 8), cu scopul de a scoate în evidență conturul. În linia 16 este creată o pensulă albastră, selectată în cadrul contextului dispozitiv în linia 19. Dreptunghiul `CRect` este micșorat în linia 21 pentru a desena un dreptunghi rotunjit (linia 22), colțurile acestuia fiind modelate de o elipsă având lățimea de 15 pixeli.

În cazul în care compilați și rulați programul astfel modificat, ar trebui să vedeți un dreptunghi rotunjit având interiorul albastru și un contur gros de culoare roșie, totul pe un fond având culoarea suprafeței de lucru.

Desenarea de cercuri și elipse umplute

Ați întâlnit deja elipse atunci când ați generat fundalul ferestrei, în listingul 16.15. Funcția `Ellipse()` pe care ați folosit-o împreună cu o peniță lucrează la fel de bine și cu pensulele, dar în acea ipostază a fost selectată o pensulă nulă pentru a împiedica generarea interiorului. Pentru a ilustra această funcție, puteți adăuga după apelul `RoundRect()` liniile de cod următoare:

```
// Micsoram dreptunghiul inca putin
rcClient.DeflateRect(25,25);
// Desenam in cadrul sau o elipsa
pDC->Ellipse(rcClient);
```

Un pont util pentru crearea de elipse cu un centru dat

De multe ori veți dori să creați o elipsă având diametre și centru bine determinate. O soluție rapidă este să creați un obiect `CRect` folosind pe post de coordonate ale colțurilor stânga-sus și dreapta-jos același obiect `CPoint` corespunzător centrului dorit. De exemplu, `CRect rcEllipse (ptCenter, ptCenter);`. Apoi veți folosi funcția `InflateRect()` pentru a stabili diametrele elipsei și veți transmite dreptunghiul obținut în apelul funcției `Ellipse()`.

După compilarea și rularea programului modificat, în interiorul dreptunghiului rotunjit ar trebui să fie afișată o elipsă mai mică. Aceasta va avea un contur roșu, care se datorează peniței selectate.

Desenarea de suprafețe de coardă și rază

O *suprafață de coardă* provine dintr-o elipsă care a fost tăiată în două printr-o linie dreaptă; fiecare dintre cele două părți este acum o suprafață de coardă. Exact așa vom folosi și funcția `Chord()`. Este necesar un obiect dreptunghi (`CRect`) care furnizează coordonatele de încadrare, ca în cazul unei elipse. Apoi se transmit două perechi de coordonate care reprezintă punctele de început și de sfârșit ale unei linii imaginare ce taie cele două bucăți. Secțiunea desenată depinde de care dintre aceste puncte este precizat mai întâi.

Operați modificările ilustrate în listingul 16.17 între selecțiile de pensulă din funcția `OnDraw()`.

LISTINGUL 16.17 LST17_17.CPP – Desenarea de suprafețe de coardă cu `Chord()`

```
1 pOldBrush = pDC->SelectObject(&brBlue);
2
3 CRect rcChord = rcClient;
4 rcChord.DeflateRect(50,50);
5
```

```
6 pDC->Chord(rcChord, — ❶
7     rcClient.TopLeft(),
8     rcClient.BottomRight());
9
10 pDC->SelectObject(pOldBrush);
```

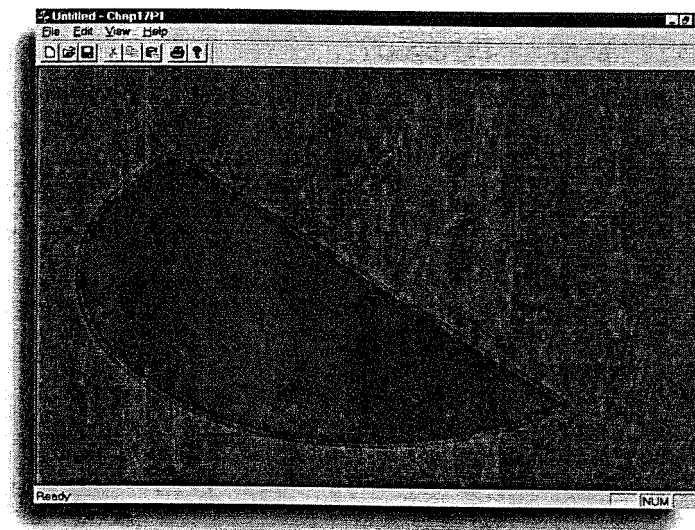
❶ Suprafața desenată rezultă prin intersectarea cu o linie trasată între colțurile stânga-sus și dreapta-jos.

Primul lucru pe care-l puteți remarca este că declarația obiectului `rcChord` din linia 3 este inițializată tot pe baza dreptunghiului client. Acest dreptunghi este micșorat cu 50 de pixeli prin apelul `Deflate()` din linia 4, fiind folosit în funcția de trasare a suprafeței de coardă din liniile 5, 6 și 7. Segmentul de divizare își are capetele în colțurile stânga-sus și dreapta-jos ale dreptunghiului client. Rezultă astfel o linie diagonală care înjumătățește elipsa dinspre colțul din stânga sus al ferestrei până înspre colțul dreapta-jos.

În cazul în care compilați și rulați programul astfel modificat, ar trebui să obțineți o suprafață de coardă, asemeni celei din figura 16.8.

FIGURA 16.8

O suprafață de coardă umplută.



Desenarea de poligoane

Funcția `Polygon()` primește același tip de parametri ca și funcțiile `Polyline()` și `PolyBezier()`. Este nevoie de o mulțime de pereche de coordonate pentru fiecare punct și de numărul acestor puncte. Spre deosebire de `Polyline()`, nu mai trebuie ca ultimul punct să fie același cu primul pentru a obține o figură închisă, pentru că poligonul este închis automat.

Există o problemă de care trebuie să țineți seama – modul de umplere. O funcție numită `SetPolyFillMode()` vă permite să alegeți între `ALTERNATE` și `WINDING`. Dacă optați pentru

ALTERNATE, poligonul este umplut așa cum apar liniile într-o parcurgere de la stânga la dreapta. Modul WINDING ține cont de direcția liniei relativ la predecesora sa.

Determinarea modului curent de umplere a poligoanelor

Modul curent de umplere a poligoanelor poate fi determinat prin apelul funcției `GetPolyFillMode()` a contextului dispozitiv. Aceasta va întoarce una dintre valorile ALTERNATE sau WINDING, în funcție de modul stabilit curent în respectivul context dispozitiv.

Operați în funcția `OnDraw()` modificările ilustrate în listingul 16.18 pentru a desena două poligoane.

LISTINGUL 16.18 LST17_18.CPP – Desenarea unor stele cu ajutorul funcției `Polygon()`

```
1 #include "math.h"
2 void CMyDrawView::OnDraw(CDC* pDC)
3 {
4     CMyDrawDoc* pDoc = GetDocument();
5     ASSERT_VALID(pDoc);
6
7     CPen penRed(PS_SOLID, 5, RGB(255, 0, 0));
8     CPen *pOldPen = NULL;
9
10    pOldPen = pDC->SelectObject(&penRed);
11
12    CRect rcClient;
13    GetClientRect(&rcClient);
14
15    CBrush brBlue(RGB(0, 0, 255));
16    CBrush *pOldBrush = NULL;
17
18    pOldBrush = pDC->SelectObject(&brBlue);
19
20    for(int w=0; w<2; w++)
21    {
22        const int nPoints = 5;
23        double nAngle = (720.0/57.295)/(double)nPoints;
24        int xOffset = (w ? 1 : -1)*rcClient.Width()/4;
25        pDC->SetPolyFillMode(w ? ALTERNATE : WINDING);
26        CPoint ptPolyAr[nPoints];
27        for(int i=0; i<nPoints; i++)
28        {
29            ptPolyAr[i].x = xOffset +
30                (long)(sin((double)i * nAngle) * 100.0);
31            ptPolyAr[i].y =
32                (long)(cos((double)i * nAngle) * 100.0);
```

```
33         ptPolyAr[i] += rcClient.CenterPoint();
34     }
35     pDC->Polygon(ptPolyAr, nPoints);
36
37 }
38
39 pDC->SelectObject(pOldBrush);
40 pDC->SelectObject(pOldPen);
41 }
```

- ❶ La prima iterație `w` este nul, deci este folosit modul WINDING. La a doua iterație, steaua este desenată cu ajutorul modului ALTERNATE.
- ❷ Vârfurile stelei sunt calculate și stocate în vectorul de puncte `ptPolyAr`.
- ❸ Poligonul poate fi desenat pe baza vectorului de puncte și a numărului de elemente ale acestuia, prin apelul funcției `Polygon()`.

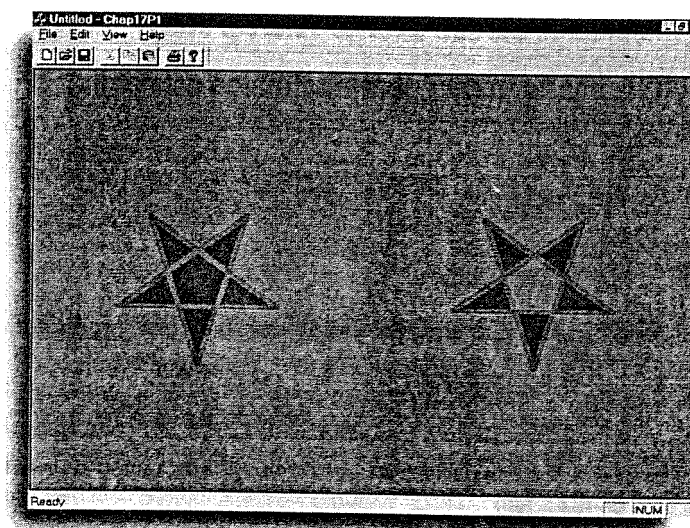
Pentru a putea utiliza funcțiile `sin` și `cos` este necesară adăugarea la început a liniei `#include "math.h"`.

În vectorul de puncte declarat în linia 26 sunt generate două stele, desenate prin apelul funcției `Polygon()` din linia 35. Prima, cea din stânga, folosește modul WINDING, în timp ce a doua folosește modul ALTERNATE. Pentru că segmentele care formează figura de stea se intersectează între ele, în modul WINDING este umplută întreaga figură, în timp ce în modul ALTERNATE rămâne goală zona din mijlocul figurii.

După compilare și rulare, poligoanele generate vor arăta ca în figura 16.9.

FIGURA 16.9

Poligoane desenate în modurile ALTERNATE și WINDING.



Afișarea de text cu diferite culori și stiluri

Utilizarea fonturilor pentru rafinarea aspectului aplicațiilor

Din momentul în care William Caxton a inventat tipărița, cuvântul tipărit a devenit omniprezent în existența noastră. Fonturile create inițial pentru tipografie s-au înmulțit și sunt disponibile într-o gamă foarte largă. Alegerea corectă a fonturilor poate avea un efect considerabil asupra aspectului și stilului unei aplicații. Windows oferă un suport extins pentru afișarea textului cu o varietate de fonturi și stiluri.

Funcții pentru afișarea textului

Unul dintre scopurile principale urmărite de cei ce au proiectat interfața grafică din Windows (GDI) a fost să ofere un mecanism universal care să permită o afișare grafică de bună calitate pentru o varietate de dispozitive. *WYSIWYG* (What You See Is What You Get – ceea ce vezi este ceea ce obții) a fost laitmotivul anilor 80' – acum este pur și simplu normal. Producerea de text cu o multitudine de fonturi și pe diferite ecrane și imprimante este unul dintre aspectele cele mai delicate în ceea ce privește caracterul general. Din fericire, proiectul a reușit și, prin intermediul unei colecții de funcții puternice pentru selectarea fonturilor și afișarea textului, aveți posibilitatea de a produce text păstrând certitudinea că acesta va fi reprezentat cu acuratețe pe dispozitive diferite.

VEDEȚI...

- Pentru mai multe informații privind contextele dispozitiv, vedeți pagina 323.
- Pentru determinarea caracteristicilor unui dispozitiv, vedeți pagina 342.

Afișarea de text simplu

Funcția membru `TextOut()` a contextului dispozitiv reprezintă probabil metoda cea mai rapidă și mai ușoară pentru afișarea de text. Parametrii necesari sunt coordonatele pe orizontală și pe verticală ale poziției de afișare și un obiect `CString` conținând textul de afișat.

Alte funcții `TextOut()`

Funcția `TextOut()` este însoțită de alte câteva funcții similare. `TabbedTextOut()` poate să dezvolte caracterele Tab la o serie de poziții de tabulare specificate într-un vector. `PolyTextOut()` poate să afișeze printr-un singur apel un vector de șiruri de caractere. `ExtTextOut()` vă permite să specificați opțiuni de generare a textului cu scopul de a afișa reprezentarea acestuia.

Puteți să folosiți și de această dată o reprezentare SDI pe post de planșă de desen. Creați o infrastructură SDI cu ajutorul AppWizard, acceptând toate opțiunile propuse implicit, așa cum v-a arătat capitolul 12, „Utilizarea documentelor, a reprezentărilor și a cadrelor”. Această aplicație SDI poate fi utilizată pentru a experimenta câteva din funcțiile de afișare a textului.

Selectați pagina ClassView a secțiunii spațiului de lucru al proiectului. Veți vedea clasa `CSimpTextView`. Efectuați un clic pe semnul plus (+) pentru a afișa membrii acestei clase. Efectuați un dublu clic pe `OnDraw()` și modificați implementarea standard în concordanță cu listingul 17.1. Noua implementare va afișa o linie de text simplu.

LISTINGUL 17.1 LST18_1.CPP – Afisarea de text cu *TextOut()*

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     CString strText = "Water under the bridge.";
9     pDC->TextOut(100,100,strText); — ❶
10 }

```

❶ *TextOut()* afișează textul înscris în variabilă la 100 de pixeli pe orizontală și pe verticală.

În listingul 17.1, textul este înscris în șirul *strText* și transmis apoi pentru afișare funcției *TextOut()*. *TextOut()* necesită și coordonatele unei poziții, specificate în linia 9 la 100 de pixeli distanță de laturile superioară și din stânga. În cazul în care compilați și rulați programul modificat, veți vedea că textul este afișat cu fontul și culoarea implicite (negru pe alb), aproape de colțul stânga sus al ferestrei.

Stabilirea alinierii textului

Textul poate fi aliniat în mai multe feluri relativ la poziția specificată, apelând în acest scop funcția *SetTextAlign()*. Acest membru al contextului dispozitiv primește o combinație de indicatori care precizează alinierea textului și modul în care poziția cursorului va fi actualizată după afișarea acestuia.

Determinarea opțiunilor curente de aliniere a textului

Opțiunile curente de aliniere a textului pentru un context dispozitiv pot fi determinate pe baza valorii întoarse de apelul funcției *GetTextAlign()*. Această valoare reprezintă o combinație formată din aceiași indicatori care pot fi transmiși funcției *SetTextAlign()*.

Indicatorii se referă la unde va fi situat punctul specificat în raport cu textul; pentru început s-ar putea să vă deruteze. V-ați aștepta ca *TA_RIGHT* să alinieze textul la dreapta punctului, numai că textul va fi aliniat la stânga, cu punctul aflat în dreapta sa. Tabelul 17.1 prezintă indicatorii care pot fi transmiși funcției *SetTextAlign()*.

TABELUL 17.1 Indicatori de aliniere a textului pentru funcția *SetTextAlign()*

Indicator de aliniere	Efectul asupra afișării textului
<i>TA_LEFT</i>	Textul este aliniat la dreapta punctului.
<i>TA_RIGHT</i>	Textul este aliniat la stânga punctului.
<i>TA_CENTER</i>	Textul este aliniat cu punctul în centru.
<i>TA_TOP</i>	Textul este aliniat dedesubtul punctului.

Indicator de aliniere

Efectul asupra afișării textului

<i>TA_BOTTOM</i>	Textul este aliniat deasupra punctului.
<i>TA_BASELINE</i>	Textul este aliniat astfel încât linia sa de bază să conțină punctul.
<i>TA_UPDATECP</i>	Poziția cursorului grafic este actualizată după fiecare apel <i>TextOut()</i> , coordonatele specificate sunt ignorate și textul este afișat imediat după poziția anterioară.
<i>TA_NOUPDATECP</i>	Poziția cursorului grafic nu este actualizată de apelurile <i>TextOut()</i> .

Puteți testa efectele opțiunilor de aliniere, desenând o cruce cu centrul în (200,200) și afișând în acest punct texte cu diferite alinieri, transmițând diverși indicatori de aliniere. Pentru a experimenta aceste opțiuni de aliniere, modificați implementarea *OnDraw()* în concordanță cu listingul 17.2.

LISTINGUL 17.2 LST18_2.CPP – Stabilirea alinierii textului

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // ** Aliniem punctul la dreapta textului
9     pDC->SetTextAlign(TA_RIGHT); — ❶
10
11     // ** Afisam un text
12     pDC->TextOut(200,200,"<-Right->");
13
14     // ** Aliniem punctul la stanga textului
15     pDC->SetTextAlign(TA_LEFT);
16     pDC->TextOut(200,200,"<-Left->");
17
18     // ** Aliniem punctul in centrul si la baza textului
19     pDC->SetTextAlign(TA_CENTER + TA_BOTTOM); — ❷
20     pDC->TextOut(200,200,"<-Center->");
21
22     pDC->MoveTo(150,200); // ** Deplasam cursorul
23     pDC->LineTo(250,200); // ** Desenam linia orizontala
24     pDC->MoveTo(200,150); // ** Deplasam cursorul
25     pDC->LineTo(200,250); // ** Desenam linia verticala
26
27 }

```

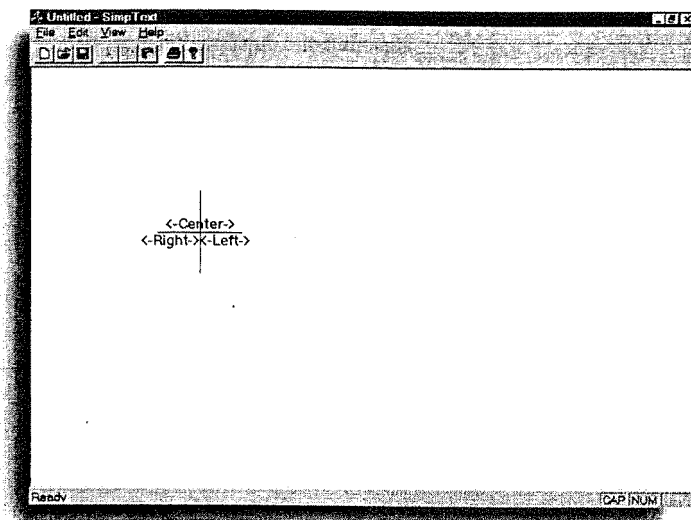
❶ Textul este aliniat la dreapta în mod implicit, dar aici modificăm această aliniere.

❷ În unele cazuri este posibilă combinarea indicatorilor. Aici, textul este centrat și plasat deasupra poziției specificate.

În fiecare apel `TextOut()` sunt transmise aceleași coordonate, dar alinierea diferă de fiecare dată, astfel că textul apare în diferite poziții față de centrul crucii. După compilarea și rularea programului veți vedea textul afișat în diverse poziții, ca în figura 17.1.

FIGURA 17.1

Diferite opțiuni de aliniere a textului.



VEDEȚI...

➤ Pentru amănunte privind desenarea de linii și figuri, vedeți pagina 355.

Modificarea culorilor de afișare și de fundal pentru text

În mod implicit, textul este afișat cu negru pe alb, dar aceste culori pot fi modificate prin intermediul funcțiilor `SetTextColor()` și `SetBkColor()` ale contextului dispozitiv.

Ambele funcții primesc un singur parametru: o referință de culoare (`COLORREF`) pentru culoarea de text sau de fundal dorită. Valoarea întoarsă este tot `COLORREF`, indicând culoarea de text sau de fundal selectată anterior.

Determinarea culorilor curente pentru afișarea textului

Culorile folosite curent la afișarea textului pot fi determinate cu ajutorul funcțiilor `GetTextColor()` și `GetBkColor()`, care întorc culorile de afișare și de fundal selectate curent în contextul dispozitiv. Ambele funcții furnizează culorile sub formă de valori `COLORREF`.

Puteți să colorați cele afișate de funcția `OnDraw()`, apelând funcțiile `SetTextColor()` și `SetBkColor()` înainte de a utiliza funcția `TextOut()`. Aceste modificări sunt prezentate în listingul 17.3.

LISTINGUL 17.3 LST18_3.CPP – Selectarea culorii de afișare a textului prin intermediul funcțiilor `SetTextColor()` și `SetBkColor()`

```
1 // ** Alegem culoarea albastra pentru text
2 pDC->SetTextColor( RGB(0,0,255) );
3 pDC->TextOut( 200,200, "<-Right->" );
4
5 pDC->SetTextAlign( TA_LEFT );
6
7 // ** Alegem culoarea rosie pentru text
8 pDC->SetTextColor( RGB(255,0,0) ); — ❶
9 pDC->TextOut( 200,200, "<-Left->" );
10
11 // Aliniam punctul in centrul si la baza textului
12 pDC->SetTextAlign( TA_CENTER + TA_BOTTOM );
13
14 // ** Alegem culoarea galbena pentru text
15 pDC->SetTextColor( RGB(255,255,0) );
16
17 // ** Alegem culoarea albastru inchis pentru fundal
18 pDC->SetBkColor( RGB(0,0,128) ); — ❷
19 pDC->TextOut( 200,200, "<-Center->" );
```

❶ Această linie stabilește culoarea cu care este afișat textul.

❷ Această linie stabilește culoarea de fundal pe care va fi afișat textul.

În listingul 17.3, apelurile `SetTextColor()` stabilesc afișarea textului cu albastru în linia 2, cu roșu în linia 8 și cu galben în linia 15. În linia 18 se alege culoarea albastru închis pentru fundal.

În cazul în care compilați și rulați programul după efectuarea acestor modificări, veți vedea același text ca mai devreme, dar afișat cu alte culori și pe alt fundal.

VEDEȚI...

➤ Pentru mai multe detalii privind referințele de culoare și macrodefiniția `RGB`, vedeți pagina 350.

Afișarea de text opac și transparent

În unele situații veți dori ca textul afișat să acopere zona aflată dedesubt. Alteori veți dori ca textul să se integreze în imaginea afișată. `SetBkMode()` vă permite să optați pentru una dintre aceste posibilități. Funcția primește un singur parametru, unul dintre indicatorii `OPAQUE` (valoarea implicită) sau `TRANSPARENT` (pentru text transparent), prin care se precizează modul dorit. Dacă apelați `SetBkMode()` cu `OPAQUE`, la generarea textului se vor folosi atât culoarea de afișare, cât și culoarea de fundal. Dacă transmiteți valoarea `TRANSPARENT`, se va folosi numai culoarea de afișare a textului.

Puteți să modificați `OnDraw()` pentru a testa modurile de afișare transparentă și opacă, în concordanță cu listingul 17.4.

LISTINGUL 17.4 LST18_4.CPP – Stabilirea modurilor de afișare opacă și transparentă a textului

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     pDC->Ellipse(160,160,240,240);
9     pDC->SetTextAlign(TA_CENTER);
10
11     // ** Selectam modul transparent
12     pDC->SetBkMode(TRANSPARENT);
13
14     // ** Afisam un text
15     pDC->TextOut(200, 180, "Transparent"); — ❶
16
17     // ** Selectam modul opac
18     pDC->SetBkMode(OPAQUE); — ❷
19
20     // ** Afisam un text
21     pDC->TextOut(200, 200, "Opaque");
22 }

```

❶ În modul transparent, cercul este vizibil sub literele textului.

❷ În modul opac, textul este afișat împreună cu un fundal care acoperă cercul.

În listingul 17.4, orice text afișat după linia 12 va avea un fundal transparent. Modul OPAQUE stabilit în linia 18 va duce la afișarea textului pe un fundal colorat opac.

În cazul în care compilați și rulați programul după efectuarea modificărilor, veți vedea că elipsa este acoperită de către cuvântul Opaque. În schimb, rămâne vizibilă sub cuvântul Transparent. Dacă alegeți o culoare de fundal, aceasta nu este folosită la afișarea în modul transparent.

Decuparea textului la un dreptunghi

De multe ori este nevoie să *decupați* textul astfel încât să nu depășească suprafața unui dreptunghi de încadrare. Chiar dacă sunt tăiate bucăți de text, soluția este uneori este de preferat față de suprapunerea textului peste restul imaginii.

ExtTextOut() este o variantă mai sofisticată a funcției TextOut(). Această funcție poate să efectueze decuparea textului prin specificarea în apel a indicatorului ETO_CLIPPED. Primii doi parametri ai funcției ExtTextOut() reprezintă o pereche de coordonate, asemenea funcției TextOut(). Prin intermediul celui de-al treilea parametru poate fi transmisă o combinație de indicatori, printre care valoarea ETO_CLIPPED are ca efect decuparea

textului. Puteți, de asemenea, să includeți indicatorul ETO_OPAQUE, care vă permite să afișați text opac chiar dacă anterior a fost selectat modul transparent. Dacă nu doriți nici decupare, nici text opac, transmiteți zero prin acest al treilea parametru (ca în linia 13 a listingului 17.5). Al patrulea parametru este un obiect CRect și, în cazul în care a fost specificat indicatorul pentru decupare, textul va fi decupat la acest dreptunghi. Textul care trebuie afișat se transmite prin al cincilea parametru (sub forma unui obiect CString). În fine, al șaselea parametru permite specificarea unui vector pentru spațierea caracterelor, spațierea implicită obținându-se prin transmiterea valorii NULL.

Indicatorul TA_UPDATECP și apelul ExtTextOut()

Indicatorul de aliniere TA_UPDATECP, specificat cu ajutorul funcției SetTextAlign(), poate fi utilizat împreună cu funcția ExtTextOut(). Atunci când acest indicator este stabilit, coordonatele transmise în primul apel TextOut() sau ExtTextOut() sunt folosite în mod normal. Următoarele apeluri ale acestor funcții, însă, ignoră coordonatele precizate și afișează textul în raport cu poziția curentă a cursorului grafic.

Pentru a efectua decuparea unui text, modificați funcția OnDraw() ca în listingul 17.5.

LISTINGUL 17.5 LST18_5.CPP – Decuparea textului cu ExtTextOut()

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     CRect rcClipBox(CPoint(100,100),CPoint(250,120));
9     pDC->Rectangle(rcClipBox);
10    pDC->SetBkMode(TRANSPARENT);
11
12    // ** Afisam textul fara decupare
13    pDC->ExtTextOut(100,100,0,rcClipBox,
14        _"This text won't fit in there!",NULL);
15
16    rcClipBox.OffsetRect(0,40);
17    pDC->Rectangle(rcClipBox);
18
19    // ** Afisam textul cu decupare
20    pDC->ExtTextOut(100,140,ETO_CLIPPED, rcClipBox, — ❶
21        _"This text won't fit in there!",NULL);
22 }

```

❶ Funcția ExtTextOut() decupează textul la dreptunghiul specificat atunci când este transmis indicatorul ETO_CLIPPED.

Linia 8 din listingul 17.5 declară un obiect dreptunghi. Acesta va fi utilizat atât pentru poziționarea, cât și pentru decuparea textului. În apelul ExtTextOut() din linia 13 nu se

specifică indicatorul `ETO_CLIPPED`, astfel că textul va depăși suprafața dreptunghiului. Apelul `ExtTextOut()` din linia 20 determină decuparea, iar textul care depășește marginile dreptunghiului nu va fi afișat.

În cazul în care compilați și rulați programul modificat în concordanță cu listingul 17.5, veți vedea două dreptunghiuri, cu text care depășește laturile unuia, dar este decupat de laturile celuilalt.

VEDEȚI...

➤ Pentru detalii privind desenarea și decuparea, vedeți pagina 330.

Crearea de fonturi

La afișarea de text ați folosit până acum fontul implicit din contextul dispozitiv. Asemeni pensulelor și penițelor, fonturile reprezintă și ele obiecte GDI și pot fi selectate în cadrul unui context dispozitiv. Fonturile pot fi create și personalizate după nevoie, prin intermediul unei serii de funcții oferite de Windows GDI și a încapsulărilor MFC.

Utilizarea clasei *CFont*

CFont este clasa care încapsulează obiectul GDI font. Spre deosebire de alte clase de încapsulare, *CFont* nu poate fi construită pur și simplu pe baza unor valori implicite și apoi folosită. În schimb, trebuie să declarați un obiect *CFont* și apoi să apelați una dintre funcțiile *Create* pentru a preciza diverșii parametri posibili.

Funcțiile *Create* nu vă furnizează întotdeauna cu exactitate fontul specificat. De fapt, procesul de mapare de fonturi analizează cererea dumneavoastră, încearcă să găsească fontul care se potrivește cel mai bine și apoi configurează acel font astfel încât rezultatul să fie cât mai aproape de cerințe (și, de obicei, reușește destul de bine).

Crearea fonturilor cu *CreatePointFont()*

CreatePointFont() reprezintă cea mai simplă cale pentru crearea rapidă a unui font. Este nevoie de trei parametri: dimensiunea fontului în zecimi de punct, tipul de caracter (de exemplu, Arial) și, în fine, contextul dispozitiv în care veți utiliza noul font. Modificați funcția *OnDraw()* ca în listingul 17.6, utilizând funcția *CreatePointFont()* pentru a afișa un text cu un font Arial de dimensiune mare.

LISTINGUL 17.6 LST18_6.CPP – Crearea fonturilor cu *CreatePointFont()*

```
1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // ** Cream un font Arial de 36 de puncte
```

```
9     CFont fnBig;
10    fnBig.CreatePointFont(360, "Arial", pDC);
11
12    CFont* pOldFont = pDC->SelectObject(&fnBig);
13
14    pDC->TextOut(50, 50, "*** 36 Pt Arial Font ***");
15
16    pDC->SelectObject(pOldFont);
17 }
```

❶ *CreatePointFont()* reprezintă o cale rapidă și ușoară pentru a crea un font de tipul și dimensiunea dorite.

Observați că în apelul funcției *CreatePointFont()* din linia 10 este transmisă valoarea 360. Aceasta reprezintă dimensiunea exprimată în zecimi de punct. Astfel vă este oferită flexibilitate la alegerea dimensiunii de font dorite. După compilare și rulare, veți vedea textul din linia 14 afișat cu un font Arial de dimensiune mare (36 de puncte).

Crearea fonturilor cu *CreateFont()*

CreateFont() este probabil una dintre cele mai sofisticate funcții întâlnite în MFC, în bună parte datorită faptului că necesită 14 parametri, dintre care majoritatea sunt combinații complexe de indicatori. Chiar și așa, nu se pot sintetiza toate opțiunile posibile care caracterizează un font, astfel că va trebui să vă mulțumiți cu atât. Prototipul funcției *CreateFont()* este prezentat în listingul 17.7.

LISTINGUL 17.7 LST18_7.CPP – Parametrii funcției *CreateFont*

```
1 BOOL CreateFont( int nHeight, int nWidth,
2     int nEscapement, int nOrientation, int nWeight,
3     BYTE bItalic, BYTE bUnderline, BYTE cStrikeOut,
4     BYTE nCharSet, BYTE nOutPrecision,
5     BYTE nClipPrecision, BYTE nQuality,
6     BYTE nPitchAndFamily, LPCTSTR lpszFacename );
```

Stabilirea înălțimii și a lățimii fontului

Primii doi parametri ai funcției *CreateFont()* reprezintă înălțimea și lățimea fontului dorit. Primul dintre aceștia, *nHeight*, poate fi negativ sau pozitiv. Dacă specificați o valoare pozitivă, maparea de font va încerca să găsească înălțimea precizată în lista fonturilor disponibile, selectând cea mai bună potrivire bazată pe înălțimea celei de font. Dacă specificați o valoare negativă pentru *nHeight*, maparea de font va încerca să potrivească înălțimea precizată cu înălțimea de caracter a fonturilor.

Reguli pentru compararea înălțimilor

La compararea înălțimilor fonturilor, procesul de mapare va alege cel mai mare font care este mai mic decât înălțimea specificată.

Diferența dintre o *celulă* și un *caracter* constă în faptul că o celulă include intervale de spațiere deasupra și dedesubtul caracterului propriu-zis, astfel că înălțimea de caracter este înălțimea de celulă din care se scad aceste intervale. O valoare nulă va determina procesul de mapare să aleagă o valoare implicită decentă (sperați dumneavoastră).

Parametrul `nWidth` se referă la lățimea medie dorită (fonturile cu dimensionare proporțională au unele caractere mai înguste și altele mai late). Puteți transmite o valoare nulă, iar procesul de mapare va determina o valoare implicită adecvată, bazată pe înălțime și pe raportul dimensiunilor.

Stabilirea înclinării și a orientării fontului

Înclinarea și *orientarea* se referă la unghiul cu care este afișat textul normal, orizontal, față de verticală și, respectiv, la unghiul de rotație al caracterelor:

- Parametrul `nEscapement` vă permite să afișați textul înclinat, specificând unghiul față de axa orizontală exprimat în zecimi de grad. O valoare de 900 pentru acest parametru are ca rezultat un text vertical.
- Parametrul `nOrientation` reprezintă unghiul dintre linia de bază a caracterelor și axa orizontală – și de această dată, în zecimi de grad.

Stabilirea caracterelor aldine, cursive, subliniate și tăiate

Următorii patru parametri privesc grosimea de caracter (unde se pot alege stiluri aldine sau subțiri) și vă permit afișarea de caractere cursive, subliniate sau tăiate.

- Parametrul `nWeight` vă permite să modificați grosimea caracterelor, de la 0 (subțire) la 1000 (gros). Există câteva valori predefinite pe care le puteți folosi, prezentate în tabelul 17.2.

TABELUL 17.2 Indicatori pentru grosimea de caracter

Indicator	Valoarea grosimii
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_HEAVY	900

Indicatori echivalenți pentru grosimea de caracter

S-ar putea să întâlniți și alți indicatori, precum `FW_ULTRALIGHT`, `FW_REGULAR`, `FW_DEMIBOLD`, `FW_ULTRABOLD` sau `FW_BLACK`. Aceștia sunt echivalenți respectivi ai indicatorilor `FW_EXTRALIGHT`, `FW_NORMAL`, `FW_SEMIBOLD`, `FW_EXTRABOLD` și `FW_HEAVY`.

- Parametrul `bItalic` afișează fontul cu caractere *cursive* în cazul în care are valoarea `TRUE`.
- Parametrul `bUnderline` afișează fontul cu caractere subliniate în cazul în care are valoarea `TRUE`.
- Parametrul `bStrikeOut` afișează fontul cu caractere ~~tăiate~~ în cazul în care are valoarea `TRUE`.

Stabilirea calității și a preciziei

Parametrii corespunzători calității și preciziei vă permit să stabiliți modul în care fontul este selectat și afișat în cadrul dispozitivului vizat. Există, de asemenea, posibilitatea de a specifica anumite stiluri, cum ar fi alegerea unui font cu dimensionare fixă sau proporțională, sau de a stabili prezența serifelor.

- Parametrul `nCharSet` vă permite să specificați un set de caractere. În mod normal veți transmite valoarea `ANSI_CHARSET` pentru a obține caracterele standard ANSI. În unele cazuri ați putea preciza `SYMBOL_CHARSET`, obținând simboluri în locul caracterelor obișnuite.
- Parametrul `nOutPrecision` vă permite să specificați modul în care procesul de mapare a fonturilor va alege un anumit font în detrimentul altuia. De regulă veți preciza valoarea `OUT_DEFAULT_PRECIS`, dar dacă doriți neapărat un font TrueType puteți să specificați `OUT_TT_PRECIS`.
- Parametrul `nClipPrecision` vă permite să specificați precizia la decupare. În mod normal se folosește `CLIP_DEFAULT_PRECIS`.

Alți indicatori pentru precizia decupării

Acești indicatori modifică felul în care este decupat un text în cazul în care o parte a sa depășește suprafața de decupare. Alți indicatori posibili sunt `CLIP_EMBEDDED` (ar trebui folosit exclusiv pentru fonturi încapsulate, protejate la scriere) și `CLIP_LH_ANGLES` (poate fi utilizat atunci când orientarea fontului este modificată). În codul unor programe mai vechi s-ar putea să întâlniți și alți indicatori; mulți dintre aceștia, însă, nu mai sunt folosiți și pot fi ignorați.

- Parametrul `nQuality` vă permite să stabiliți modul în care se face compromisul între calitatea aspectului caracterelor și potrivirea celorlalți parametri specificați. Valorile posibile sunt `DEFAULT_QUALITY`, `DRAFT_QUALITY` și `PROOF_QUALITY`.
- Parametrul `nPitchAndFamily` vă permite să stabiliți concomitent două elemente. Din această cauză, este foarte ușor să specificați în loc doi parametri și să obțineți o eroare la compilare pentru că `CreateFont()` nu primește 15 parametri. Dacă întâlniți un

astfel de mesaj, asigurați-vă că cele două valori sunt combinate (cu + sau |), și nu separate prin virgulă.

Opțiunile posibile pentru dimensionarea caracterelor sunt prezentate în tabelul 17.3.

TABELUL 17.3 Indicatori pentru dimensionare

Indicator pentru dimensionare	Descriere
DEFAULT_PITCH	Orice tip de font
VARIABLE_PITCH	Exclusiv fonturi cu dimensionare proporțională
FIXED_PITCH	Exclusiv fonturi cu dimensionare fixă, utile pentru emulatoarele de terminale

Indicatorii pentru dimensionare din tabelul 17.3 pot fi combinați în cadrul parametrului `nPitchAndFamily` cu indicatorii pentru familia de fonturi, prezentați în tabelul 17.4. De exemplu, pentru un font cu dimensionare oarecare și cu aspect de scris de mână ați putea să specificați combinația `DEFAULT_PITCH | FF_SCRIPT`.

TABELUL 17.4 Indicatori pentru familia de fonturi

Indicator pentru familia de fonturi	Descriere
FF_DECORATIVE	Fonturi deosebite
FF_DONTCARE	Orice font
FF_MODERN	Fonturi cu dimensionare fixă sau cu grosime constantă a liniei
FF_ROMAN	Fonturi cu dimensionare proporțională și grosime variabilă a liniei
FF_SCRIPT	Fonturi de tip scris de mână
FF_SWISS	Fonturi cu dimensionare proporțională, dar fără serife
TMPF_TRUETYPE	Este obligatorie o versiune de font TrueType

Utilizarea fonturilor TrueType

Puteți să combinați indicatorul `TMPF_TRUETYPE` cu unul dintre indicatorii pentru familia de fonturi cu scopul de a solicita un anumit membru al familiei unui font TrueType.

Fonturile *TrueType* sunt fonturi generate ca o serie de curbe și segmente. Din această cauză, fonturile TrueType își mențin calitatea la scalare mult mai bine decât fonturile de tip rastru.

Stabilirea exactă a unui nume de font

Parametrul `lpszFacename` vă permite să specificați numele unuia dintre fonturile instalate curent, ca de pildă Times New Roman.

Crearea unui font cu ajutorul funcției `CreateFont()`

Puteți să încercați un exemplu care să creeze și să utilizeze un font, modificându-i parametrii. În acest scop va trebui să modificați funcția membru `OnDraw()` a clasei reprezentare așa cum o arată listingul 17.8. Acest listing afișează o linie de text care este rotită în jurul unui punct central prin modificarea orientării pe un interval de 360 de grade.

LISTINGUL 17.8 LST18_8.CPP – Crearea fonturilor cu `CreateFont()`

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // ** Declaram un obiect CRect care va retine zona client
9     CRect rcClient;
10    GetClientRect(&rcClient);
11
12    // ** Determinam centrul dreptunghiului
13    CPoint ptCentre = rcClient.CenterPoint();
14    pDC->SetBkMode(TRANSPARENT);
15
16    // ** Parcurgem un interval de 360 de grade
17    for(int i=0 ; i < 360 ; i+=18)
18    {
19        CFont fnBig;
20
21        // ** Înălțimea 30, lățimea implicită
22        fnBig.CreateFont(30,0, — ❶
23            // ** Modificam orientarea
24            i*10,i*10,
25            // ** Sporim grosimea de caracter
26            i/4,
27            FALSE,
28            TRUE, //** Subliniat
29            FALSE,
30            ANSI_CHARSET,
31            OUT_DEFAULT_PRECIS,
32            CLIP_DEFAULT_PRECIS,
33            PROOF_QUALITY,
34            DEFAULT_PITCH + FF_DONTCARE,
35            "Arial"
36        );
37    }

```

LISTINGUL 17.8 Continuare

```

38     CFont* pOldFont = pDC->SelectObject(&fnBig);
39
40     // ** Afisam textul
41     pDC->TextOut(ptCentre.x, ptCentre.y, ".....Beautiful Fonts");
42
43
44     pDC->SelectObject(pOldFont);
45 }
46 )

```

❶ Funcția `CreateFont()` permite procesului de mapare a fonturilor să aleagă și să furnizeze un font sofisticat, dar poate fi dificil de utilizat din cauza mulțimii de parametri.

❷ Acest simplu apel `TextOut()` afișează textul cu fontul rotit.

În listingul 17.8, liniile 17-44 rotesc textul în jurul unui centru determinat în linia 13. În întinsul apel `CreateFont()` care începe în linia 22 este transmisă o valoare crescătoare a grosimii de caracter (linia 26). Unghiul curent este transmis în linia 24 sub forma parametrilor pentru orientare și înclinare, determinând afișarea textului sub toate unghiurile iterate în buclă.

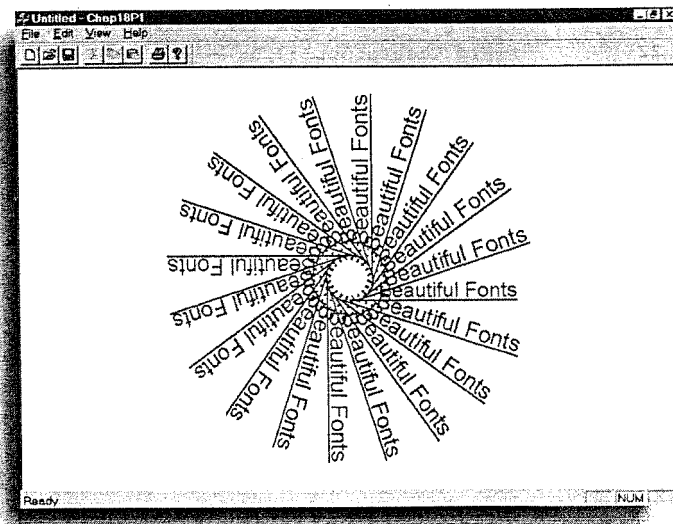
Crearea fontului o singură dată, pentru accelerarea aplicației

În cazul unei aplicații concrete, probabil că veți crea fontul o singură dată și-l veți reține ca variabilă membru a clasei reprezentare, eliminând astfel timpul pierdut cu crearea repetată a fontului.

La compilarea și rularea acestei versiuni veți vedea textul rotit în jurul centrului ferestrei, așa cum o arată figura 17.2.

FIGURA 17.2

Text rotit cu ajutorul funcției `CreateFont()`.



Selectarea și alegerea fonturilor

Utilizatorul Windows din ziua de azi este obișnuit să aleagă dintr-o gamă largă de fonturi disponibile. Pe lângă procesoarele de text, multe aplicații oferă acum această facilități în mod obișnuit. Pentru ca propriile aplicații să dispună de o asemenea flexibilitate vă este necesară o metodă de a obține lista fonturilor instalate și parametrii acestora. GDI vă oferă această posibilitate prin intermediul unor funcții de iterare a fonturilor și a unor casete de dialog speciale pentru selecția fonturilor.

Iterarea fonturilor

Cum poate un program să obțină o listă cu fonturile disponibile curent și cu toți parametrii acestora? Răspuns: Utilizând funcția `EnumFontFamilies()` și o funcție cu apel invers asociată. *toate fonturile*

Funcțiile cu apel invers se folosesc aici în felul următor: se creează o funcție care va fi apelată pentru fiecare element dintr-o listă și numele acestei funcții este transmis unui iterator. Iteratorul parcurge lista de elemente (în acest caz, fonturi) și execută funcția cu apel invers pentru fiecare dintre elemente.

Funcțiile iterator în C++

Funcțiile iterator erau obișnuite în lumea C a codului Win32. Numai că tehnicile de apel invers nu permit transmiterea contextului unui obiect, așa cum îl oferă în mod normal cuvântul cheie `this` din C++. Aceasta înseamnă că funcțiile cu apel invers trebuie să fie funcții globale, iar acestea nu se potrivesc prea bine cu modelul orientat către obiect din C++. Dar cum clasele MFC în C++ trebuie să utilizeze nucleul WIN32 asemeni oricărui program Windows (indiferent de limbajul în care este scris), sunt nevoite să accepte utilizarea funcțiilor globale cu apel invers în aceste situații particulare.

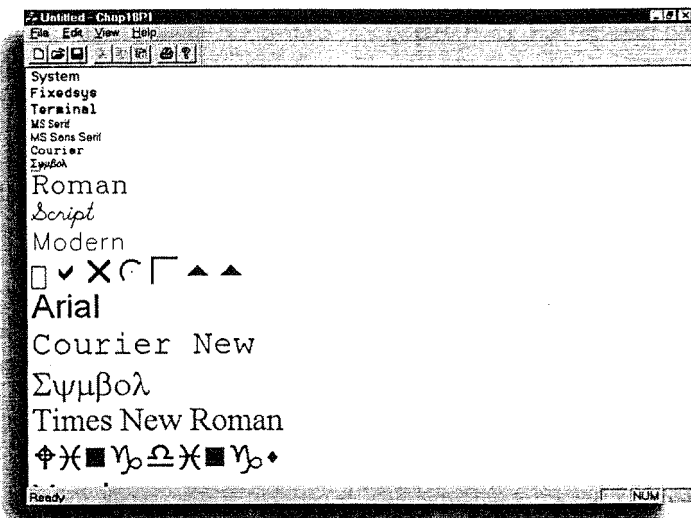
Există posibilitatea de a folosi un astfel de iterator care să parcurgă lista de fonturi și să extragă informațiile asociate fiecărui font. Toate detaliile despre un font sunt transmise apoi funcției cu apel invers.

Spre exemplu, puteți să obțineți o listă a fonturilor instalate, fiecare afișat cu propriile caractere, așa cum o ilustrează figura 17.3. Codul din listingul 17.9 și explicațiile care-i urmează vă arată modul în care puteți realiza acest lucru cu ajutorul funcției

`EnumFontFamilies()`.

Pentru a putea afișa lista de fonturi (înfățișată în figura 17.3) trebuie să adăugați înainte de `OnDraw()` funcția cu apel invers. Observați că această funcție este globală și, prin urmare, numele său nu este precedat de nici un identificator de clasă. Funcția trebuie plasată anterior implementării `OnDraw()`; altfel ați fi nevoit să adăugați o declarație corespunzătoare a funcției înainte de `OnDraw()`. În ceea ce privește funcția `OnDraw()`, în cadrul său veți adăuga apelul `EnumFontFamilies()`, ca în listingul 17.9.

FIGURA 17.3
Fonturi iterate cu
EnumFontFamilies(
).



LISTINGUL 17.9 LST18_9.CPP – Iterarea și afișarea fonturilor instalate

```

1 // ** Funcția cu apel invers pentru fonturi
2 int CALLBACK FontCallBack(ENUMLOGFONT FAR* lpelf, — ❶
3                             NEWTEXTMETRIC FAR *lpnt,
4                             int FontType,
5                             LPARAM lParam)
6 {
7     // ** lParam ne furnizează un pointer la contextul dispozitiv
8     CDC* pDC = (CDC*)lParam;
9     CFont fnEnum;
10
11     // ** Cream fontul indirect
12     fnEnum.CreateFontIndirect(&lpelf->elfLogFont); — ❷
13
14     CFont* pOldFont = pDC->SelectObject(&fnEnum);
15
16     // ** Determinăm poziția curentă a cursorului
17     int nYPos = pDC->GetCurrentPosition().y;
18
19     // ** Afișăm numele fontului
20     pDC->TextOut(5, nYPos, CString(lpelf->elfLogFont.
21                                     lfFaceName));
22
23     // ** Coborâm cu înălțimea fontului
24     pDC->MoveTo(0, nYPos + lpelf->elfLogFont.lfHeight); — ❸

```

```

25     pDC->SelectObject(pOldFont);
26
27     // ** întoarcem TRUE pentru a continua iterarea
28     return TRUE;
29 }
30
31
32
33 ///////////////////////////////////////////////////
34 // CSimpTextView drawing
35
36 void CSimpTextView::OnDraw(CDC* pDC)
37 {
38     CSimpTextDoc* pDoc = GetDocument();
39     ASSERT_VALID(pDoc);
40
41     // TODO: add draw code for native data here
42
43     // ** Apelăm iteratorul, transmitându-i funcția cu apel invers
44     EnumFontFamilies(pDC->GetSafeHdc(), NULL, — ❹
45                     (FONTENUMPROC) FontCallBack, (LPARAM) pDC);
46 }

```

- ❶ Funcția cu apel invers va fi executată pentru fiecare dintre fonturile instalate în sistem. Detaliile asociate acestora sunt transmise ca parametri.
- ❷ Funcția CreateFontIndirect() este similară cu funcția CreateFont(), dar parametrii sunt transmiși ca membri ai unei structuri LOGFONT.
- ❸ Funcția MoveTo() este apelată aici cu scopul de a pregăti cursorul grafic pentru afișarea următorului font.
- ❹ Adresa funcției cu apel invers este transmisă funcției EnumFontFamilies() astfel încât aceasta să o poată apela pentru fiecare dintre fonturile instalate.

În listingul 17.9, prima funcție prezentată (între liniile 1-29) este funcția cu apel invers. Această funcție va fi apelată pentru fiecare dintre fonturile instalate în sistem, cu detaliile asociate fontului transmise ca parametri. Așadar, ce se întâmplă aici? În OnDraw() este apelat iteratorul care folosește apelul invers.

Parametrul lParam este o valoare definită de utilizator, care în cazul nostru reprezintă un pointer la contextul dispozitiv, specificat în apelul iteratorului din linia 44. Acest context dispozitiv va fi utilizat ulterior pentru afișarea fiecăruia dintre diferitele fonturi compatibile cu dispozitivul respectiv (în acest caz, dispozitivul este ecranul).

Funcția CreateFontIndirect() din linia 12 este similară cu CreateFont(), dar primește o structură care conține toți parametrii, așa cum veți vedea mai târziu în listingul 17.11.

Al doilea parametru transmis funcției iterator în linia 44 este familia de fonturi. Prin specificarea valorii NULL vor fi considerate toate familiile de fonturi. Ați fi putut să

transmiteți o familie anume, precum Arial sau Courier, și ați fi obținut doar fonturile din acea familie.

Al treilea parametru este numele funcției cu apel invers, convertit la (FONTENUMPROC) pentru a evita proteste din partea compilatorului.

În fine, prin ultimul parametru puteți să transmiteți o informație proprie. Pentru că intenționați să afișați fonturile, contextul dispozitiv (în cadrul căruia veți afișa) este probabil cel mai util de transmis.

Precum vedeți, iteratorul găsește lista fonturilor instalate și începe să apeleze funcția cu apel invers `FontCallback()`, transmițându-i mai mulți parametri. Primul dintre aceștia este o structură `ENUMLOGFONT`. Listingul 17.10 prezintă definiția acestei structuri.

LISTINGUL 17.10 LST18_10.CPP – Structura `ENUMLOGFONT`

```
1 typedef struct tagENUMLOGFONT {
2     LOGFONT    elfLogFont;
3     BCHAR      elfFullName[LF_FULLFACESIZE];
4     BCHAR      elfStyle[LF_FACESIZE];
5 } ENUMLOGFONT;
```

Tipul de date `BCHAR`

S-ar putea să întâlniți tipul de date `BCHAR` în structurile `WIN32`. Acest tip de date este definit ca `BYTE` atunci când programul este compilat într-un mediu obișnuit și ca `WCHAR` (doi octeți) într-un mediu `Unicode`.

`Unicode` reprezintă fiecare caracter printr-o secvență de doi octeți, ceea ce face posibilă existența unor seturi de 65.536 de caractere, permițând astfel includerea caracterelor și simbolurilor diferitelor alfabeturi într-un set de caractere unic.

Structura `ENUMLOGFONT` transmisă conține numele `elfFullName` și stilul `elfStyle` al fontului, dar probabil că cel mai util membru este structura `LOGFONT`. Aceasta este o altă versiune pentru mulțimea parametrilor care se transmit funcției `CreateFont()`. Așa cum ați văzut în linia 12 a listingului 17.9, structura `LOGFONT` poate fi transmisă funcției `CreateFontIndirect()`. Listingul 17.11 prezintă definiția acestei structuri din `Windows`.

LISTINGUL 17.11 LST18_11.CPP – Structura `LOGFONT`

```
1 typedef struct tagLOGFONT { // lf
2     LONG lfHeight;
3     LONG lfWidth;
4     LONG lfEscapement;
5     LONG lfOrientation;
6     LONG lfWeight;
7     BYTE lfItalic;
8     BYTE lfUnderline;
9     BYTE lfStrikeOut;
10    BYTE lfCharSet;
```

```
11    BYTE lfOutPrecision;
12    BYTE lfClipPrecision;
13    BYTE lfQuality;
14    BYTE lfPitchAndFamily;
15    TCHAR lfFaceName[LF_FACESIZE];
16 } LOGFONT;
```

Toate aceste informații pot fi accesate prin intermediul membrului de tip `LOGFONT` al structurii `ENUMLOGFONT` către care indică pointerul `lpelf`, pointer transmis funcției cu apel invers (așa cum se vede în linia 2 a listingului 17.9). Acest pointer este folosit prima dată în linia 12 (unde construcția `&lpelf->elfLogFont` determină adresa structurii `LOGFONT`).

Următorul parametru transmis funcției cu apel invers din listingul 17.9 este o structură `NEWTEXTMETRICS`. Aceasta reține informații despre dimensiunile caracterelor și despre aspectul fontului în cadrul unui anumit context dispozitiv.

Structurile `TEXTMETRICS`

După cum probabil ați intuit, structura `NEWTEXTMETRICS` are un predecesor numit `TEXTMETRICS`. De fapt, există versiuni diferite pentru mai multe structuri `WIN32` și funcții diferite necesită versiuni ușor diferite. Confuzia în cazul nostru este sporită de faptul că există și structurile `NEWTEXTMETRICSEX`, `TEXTMETRICW` și `TEXTMETRICA`, care privesc, respectiv, semnăturile de fonturi, codificările `Unicode` și codificările `non-Unicode`.

Urmează parametrul `FontType`. Acesta vă arată dacă fontul este `TRUETYPE_FONTTYPE`, `RASTER_FONTTYPE` sau `DEVICE_FONTTYPE`. În fine, ultimul parametru vă furnizează un pointer la contextul dispozitiv, tipul parametrului fiind `LPARAM`, motiv pentru care este convertit imediat la forma cunoscută `CDC*`.

După crearea și selectarea fontului, apelul `GetCurrentPosition()` din linia 17 a listingului 17.9 determină poziția curentă a cursorului grafic din cadrul contextului dispozitiv. Numele fontului este obținut din structura `LOGFONT` (așa cum se vede în linia 20 din listingul 17.9), fiind afișat printr-un apel al funcției `TextOut()`.

Poziția cursorului grafic este deplasată în jos cu înălțimea fontului curent, prin apelul `MoveTo()` din linia 23. După ce selectați la loc fontul original trebuie să întoarceți valoarea `TRUE` pentru a semnala iteratorul să continue. Altfel, o valoare `FALSE` îi va arăta că nu doriți un nou apel.

Utilizarea casetei de dialog pentru selecția fonturilor

`Windows` oferă o casetă de dialog standard pentru selectarea unui font și a opțiunilor asociate. Multe aplicații folosesc această casetă de dialog și probabil că ați întâlnit-o de mai multe ori.

Pentru a folosi într-o aplicație caseta de dialog pentru selecția fonturilor trebuie să adăugați mai întâi o nouă opțiune de meniu care să determine apelarea casetei de dialog. În acest scop puteți să adăugați un nou element de meniu, având eticheta `Choose Font` (așa cum se vede în figura 17.4) și identificatorul `ID_CHOOSEFONT`. Secvența de pași „Inserarea unui nou element de meniu” din capitolul 13, „Lucrul cu meniuri”, vă arată ceea ce aveți de făcut.

Apoi va trebui să adăugați în clasa `CSimpTextView` o funcție care să răspundă la comanda generată de noua opțiune de meniu (`ID_CHOOSEFONT`). Această funcție se va numi `OnChooseFont()` și o puteți adăuga cu ajutorul secvenței de pași „Adăugarea unei funcții de tratare pentru o comandă de meniu cu ClassWizard”, aflată de asemenea în capitolul 13, „Lucrul cu meniuri”.

FIGURA 17.4

Inserarea unui element de meniu cu eticheta Choose Font.

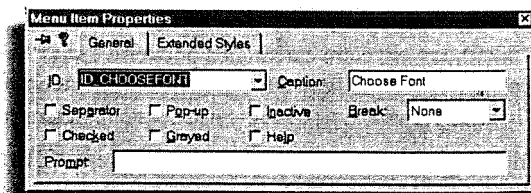
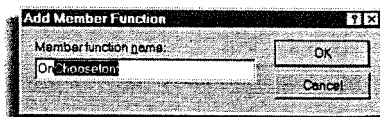


FIGURA 17.5

Adăugarea funcției de tratare a meniului, `OnChooseFont`, prin intermediul ClassWizard.



Acum aveți un element de meniu și o funcție de tratare asociată în care veți putea adăuga și folosi clasa `CFontDialog`, așa cum o ilustrează listingul 17.12.

LISTINGUL 17.12 LST18_12.CPP – Implementarea funcției `OnChooseFont()` de tratare a meniului

```
1 void CSimpTextView::OnChooseFont()
2 {
3     // TODO: Add your command handler code here
4
5     // ** Declaram un obiect CFontDialog
6     CFontDialog dlgChooseFont; ①
7     if (dlgChooseFont.DoModal() == IDOK)
8     {
9         m_fnCustom.DeleteObject();
10        m_fnCustom.CreateFontIndirect(
11            dlgChooseFont.m_cf.lpLogFont);
12        Invalidate();
13    }
14 }
```

① `CFontDialog` încapsulează caseta de dialog standard pentru selecția fonturilor, oferind o cale facilă, dar versatilă, de a permite utilizatorului să selecteze un font și un stil anume.

În listingul 17.12, linia 6 conține declarația unui obiect `dlgChooseFont` de tip `CFontDialog`. Caseta de dialog este afișată prin apelul `DoModal()` din linia 7. Utilizatorul va selecta acum noul font, iar în cazul în care efectuează clic pe **OK** va trebuie să apeleți

funcția `DeleteObject()` pentru a elibera obiectul GDI asociat fontului curent. După aceea puteți să creați noul font pe baza membrului `m_cf` al obiectului `CFontDialog`, membru care reprezintă un pointer la o structură `LOGFONT`. Apeleți apoi `Invalidate()` (linia 12) pentru a forța redesenarea ferestrei și execuția funcției `OnDraw()`.

În continuare trebuie să adăugați în clasa reprezentare o variabilă `m_fnCustom` de tip font. Efectuați un clic pe pagina **ClassView** a secțiunii spațiului de lucru al proiectului și apoi efectuați un dublu clic pe elementul `CSimpTextView` pentru a accesa declarația clasei reprezentare. Adăugați membrul dorit sub comentariul `// Attributes`, așa cum se vede în listingul 17.13.

LISTINGUL 17.13 LST18_13.CPP – Adăugarea în clasa reprezentare a unui membru de tip `CFont`

```
1 // Attributes
2 public:
3     // ** Noul membru de tip font
4     CFont m_fnCustom;
5
6     CSimpTextDoc* GetDocument();
```

În fine, trebuie să modificați implementarea `OnDraw()` în sensul afișării unui mesaj cu fontul selectat. Listingul 17.14 vă prezintă noua implementare.

LISTINGUL 17.14 LST18_14.CPP – Afișarea cu fontul selectat

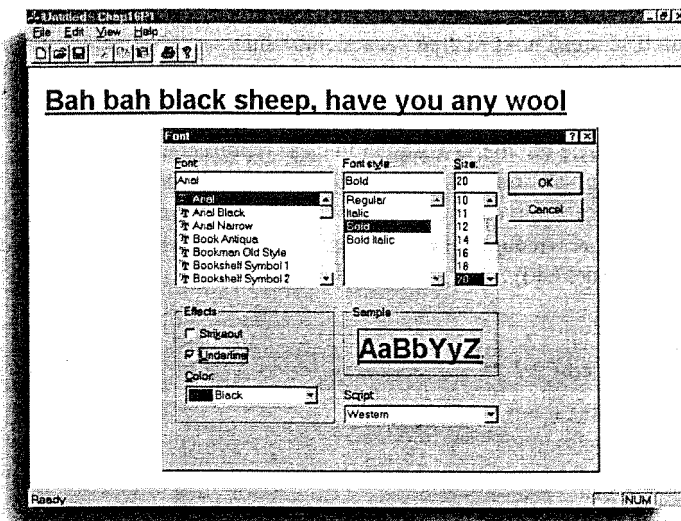
```
1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     CFont* pOldFont = pDC->SelectObject(&m_fnCustom);
9     pDC->TextOut(20,20,
10        _"Bah bah black sheep, have you any wool");
11     pDC->SelectObject(pOldFont);
12 }
```

Fontul este selectat în linia 8 a listingului 17.14, iar mesajul de test este afișat cu `TextOut()` în liniile 9 și 10.

Compilați și rulați. Dacă totul este în regulă, veți avea posibilitatea de a modifica fontul cu care este afișat mesajul prin efectuarea unui clic pe meniul **Edit** și selectarea noii opțiuni **Choose Font**. Modificați opțiunile din caseta de dialog prezentată și mesajul va fi afișat în concordanță, așa cum o ilustrează figura 17.6.

FIGURA 17.6

Caseta de dialog Font în acțiune.



VEDEȚI...

- Pentru adăugarea și implementarea diferitelor meniuri și a stilurilor asociate, vedeți pagina 270.

Afișarea de text formatat și desfășurat pe mai multe linii

Fiindcă dimensiunea fontului influențează direct mărimea suprafeței de afișare din cadrul reprezentării, scrierea de cod care să afișeze un text formatat adecvat într-o casetă bine determinată este o operație delicată. Din fericire, Windows oferă o funcție care vă ajută în acest sens. `DrawText()` primește trei parametri. Primul este un obiect `CString` care conține textul de afișat. Al doilea este dreptunghiul de încadrare a textului. Ultimul (dar sigur nu cel de pe urmă) este o combinație de indicatori care specifică exact modul de afișare și formatare a textului.

Utilizarea funcției `DrawTextEx()`

Există o versiune rafinată a funcției `DrawText()`, numită `DrawTextEx()`. Aceasta este disponibilă exclusiv ca funcție globală WIN32, așa că va trebui să transmiteți ca prim parametru indicatorul asociat contextului dispozitiv destinație. Această formă extinsă primește parametri obișnuiți (după indicatorul de context dispozitiv), dar permite și transmiterea ca ultim parametru a unui pointer la o structură `DRAWTEXTPARAMS`. Această structură poate fi utilizată pentru a specifica pozițiile de tabulare și marginile stângă și dreaptă.

Diversele moduri de combinare a indicatorilor au diferite efecte. Tabelul 17.5 prezintă acești indicatori.

Puteți să modificați funcția `OnDraw()` din programul care ilustrează selecția fonturilor, folosind `DrawText()` în locul funcției `TextOut()`, ca în codul prezentat de listingul 17.15.

TABELUL 17.5 Indicatori de formatare pentru `DrawText()`

Indicator	Descriere
DT_LEFT	Aliniază rândurile de text la marginea stângă a casetei.
DT_RIGHT	Aliniază rândurile de text la marginea dreaptă a casetei.
DT_CENTER	Centrează textul în cadrul casetei.
DT_VCENTER	Centrează textul pe verticală.
DT_WORDBREAK	Trece automat la linia următoare odată cu cuvântul care nu mai încapă pe linia curentă (asemeni unui procesor de text).
DT_CALCRECT	Nu afișează textul; doar calculează dreptunghiul de încadrare.
DT_EXPANDTABS	Transformă caracterele de tabulare în secvențe de spații (câte 8, în mod implicit).
DT_NOPREFIX	Nu ține cont de prefixare; altfel, caracterele & sunt tratate ca prefixe pentru taste de acces.
DT_TOP	Afișează textul în partea superioară (pe o singură linie).
DT_BOTTOM	Afișează textul în partea inferioară (pe o singură linie).

LISTINGUL 17.15 LST18_15.CPP – Afișarea de text formatat cu `DrawText()`

```

1 void CSimpTextView::OnDraw(CDC* pDC)
2 {
3     CSimpTextDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     CFont* pOldFont = pDC->SelectObject(&m_fnCustom);
9
10    CRect rcSmall(CPoint(20,20),CPoint(200,100));
11    pDC->Rectangle(rcSmall);
12    pDC->SetBkMode(TRANSPARENT);
13    // ** Apelam DrawText pentru a afisa mesajul intr-o caseta,
14    // ** desfasurat pe linii si centrat
15
16    pDC->DrawText(
17        "Bah bah black sheep, have you any wool",
18        rcSmall, DT_WORDBREAK + DT_CENTER); ❶
19
20    pDC->SelectObject(pOldFont);
21 }
```

❶ Prin transmiterea indicatorilor `DT_WORDBREAK` și `DT_CENTER` în apelul `DrawText()`, mesajul este desfășurat pe linii și centrat în cadrul dreptunghiului specificat.

În linia 16 a listingului 17.15 puteți vedea cum funcția `DrawText()` este utilizată pentru afișarea unui mesaj în interiorul dreptunghiului `rcSmall`. În cazul în care compilați și rulați programul astfel modificat, veți vedea că textul este centrat și desfășurat elegant pe linii în interiorul dreptunghiului, ca în figura 17.7. Prin selectarea proprietăților de font puteți observa efectul pe care îl are alegerea unei alte dimensiuni asupra textului din dreptunghi.

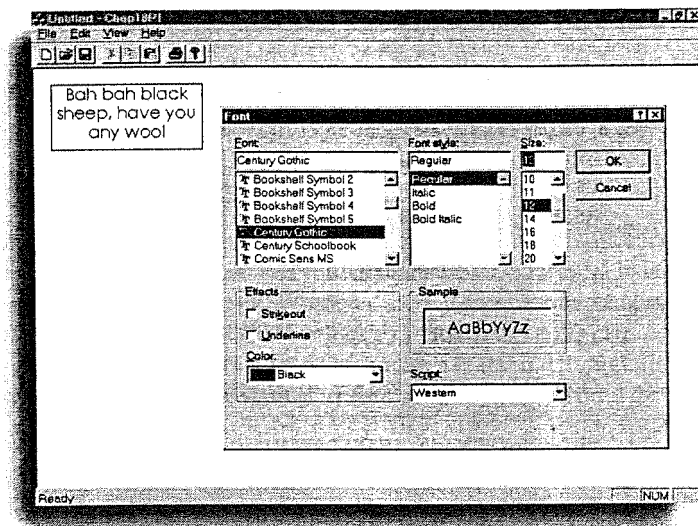
VEDEȚI...

➤ Pentru introducerea textului de către utilizator prin intermediul controalelor de editare, vedeți pagina 100.

Ștergerea fonturilor

Ca și în cazul celorlalte clase de încapsulare, fonturile GDI aflate la bază sunt șterse automat în momentul în care durata de viață a obiectelor încapsulate ia sfârșit. Puteți efectua și explicit această ștergere, apelând funcția `DeleteObject()` (cu condiția ca fontul să nu fie selectat curent în vreun context dispozitiv). În consecință, ați putea să refolosiți clasa de încapsulare apelând una dintre funcțiile `Create`.

FIGURA 17.7
Text formatat cu
`DrawText()`.



Dintre toate obiectele GDI, fonturile sunt probabil cele mai mari consumatoare de memorie, așa că merită, pe cât posibil, să ștergeți fonturile de care nu mai aveți nevoie. În cazul fonturilor utilizate mai mult, soluția cea mai bună este să le declarați ca variabile membru ale clasei reprezentare, selectându-le după nevoie și evitând astfel crearea repetată la fiecare desenare, metodă ce ar diminua performanțele programului.

VEDEȚI...

➤ Pentru mai multe amănunte privind obiectele GDI, vedeți pagina 323.

CAPITOLUL

18

Dimensionarea și derularea reprezentărilor

Sesizarea dimensionării de către utilizator a ferestrei și implementarea unui răspuns din partea programului

Stabilirea limitelor minimă și maximă pentru dimensiunile ferestrei

Afișarea de text și grafică cu dimensiuni mai mari decât ale ecranului sau ferestrei și utilizarea barelor de derulare pentru afișarea diferitelor porțiuni

Redimensionarea ferestrelor este una dintre operațiile efectuate cel mai frecvent în mediul Windows. Privim această flexibilitate ca pe ceva firesc, fără a ne gândi prea mult de obicei la implicațiile asupra aplicațiilor în cauză. Și totuși, din punctul de vedere al aplicației – sau al programatorului – redimensionarea poate fi o problemă majoră. Ar putea fi necesară modificarea pozițiilor sau dimensiunilor controalelor; s-ar putea să fie nevoie de adăugarea sau înlăturarea unor bare de derulare; grafica sau textul ar putea necesita o redesenare sau o dimensionare adecvată.

Clasele MFC ajută considerabil, dar dumneavoastră decideți modul în care aplicația va răspunde, eventual, la redimensionare.

Tratarea redimensionării ferestrei

De fiecare dată când agățați și redimensionați o fereastră se întâmplă anumite lucruri. Mai întâi, în momentul în care agățați fereastra, Windows o blochează pentru a vă împiedica să desenați pe suprafața sa. Pe măsură ce redimensionați fereastra, aceasta primește mai multe mesaje de dimensionare (`WM_SIZING`).

Determinarea stării și poziției curente ale ferestrei

Informații privind starea și poziția curente ale unei ferestre pot fi determinate în orice moment apelând funcția `GetWindowPlacement()` a obiectului `CWnd` asociat. Această funcție înscrie într-o structură `WINDOWPLACEMENT` (transmisă sub forma unui pointer ca prim parametru) detalii privind poziția curentă a ferestrei și stările de minimizare, maximizare și vizibilitate. O funcție reciprocă, `SetWindowPlacement()`, vă permite să modificați prin program starea unei ferestre.

Atunci când eliberați butonul mouse-ului, aplicația va primi un ultim mesaj (`WM_SIZE`), care arată că fereastra trebuie redimensionată la dimensiunile stabilite.

Crearea unei aplicații exemplificative

Pentru a studia aceste aspecte legate de dimensionare puteți să folosiți o aplicație SDI cu o reprezentare de tip formular. O reprezentare formular afișează pe suprafața ferestrei o machetă de dialog, ceea ce vă permite să adăugați controale în cadrul reprezentării, ca în cazul unei casete de dialog.

Secvența de pași care urmează vă arată cum puteți crea o aplicație cu o reprezentare formular prin intermediul AppWizard.

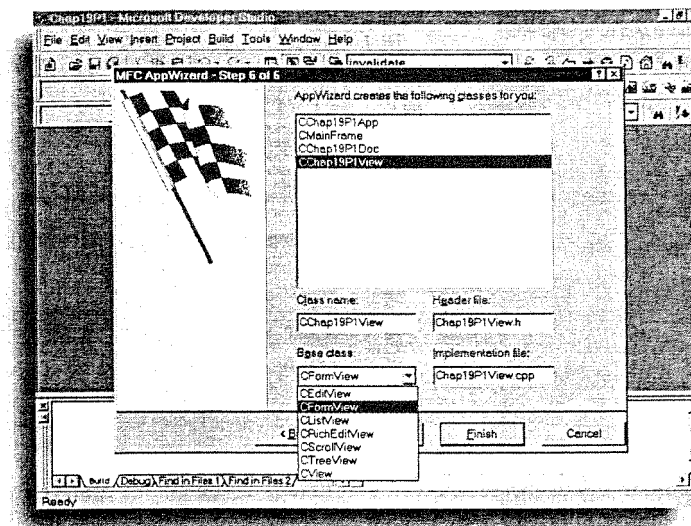
Crearea unei aplicații SDI cu o reprezentare de tip formular

1. Efectuați un clic pe meniul **File** și selectați **New**.
2. Selectați pagina **Projects** și alegeți **MFC AppWizard (exe)** din lista cu tipurile de proiecte.
3. Acum efectuați un clic pe caseta **Project Name** și introduceți numele proiectului, `SizeForm`.

4. Efectuați un clic pe **OK**. Acum ar trebui să apară caseta de dialog MFC AppWizard – Step 1.
5. Selectați **Single Document** și efectuați clic pe **Next** până când ajungeți la caseta de dialog cu steagul caroiat.
6. Efectuați un clic pe clasa `CSizeFormView` în lista claselor create de AppWizard.
7. Caseta combinată **Base Class** ar trebui să fie activată. Efectuați un clic pe săgeata de derulare pentru a afișa lista cu clasele de bază posibile pentru reprezentare.
8. Selectați `CFormView`, așa cum o ilustrează figura 18.1.
9. Efectuați un clic pe **Finish**.
10. Efectuați un clic pe **OK** în caseta de dialog New Project Information, iar AppWizard va crea noul proiect și fișierele sale sursă.

FIGURA 18.1

Selectarea clasei `CFormView` ca bază pentru reprezentare, în cadrul AppWizard.



Tratarea evenimentului de dimensionare

Ați putea să urmăriți și să interceptați mesajul de dimensionare (`WM_SIZING`) trimis de Windows pentru a afișa dimensiunile curente pe bara de titlu a ferestrei.

Fereastra este blocată pe durata operației de dimensionare, ceea ce înseamnă că va trebui, de asemenea, să o deblocați și apoi să o blocați la loc. Exemplul următor (`SizeForm`) vă arată cum puteți realiza acest lucru într-o aplicație SDI cu reprezentare formular. Adăugați o funcție de tratare `OnSizing()`.

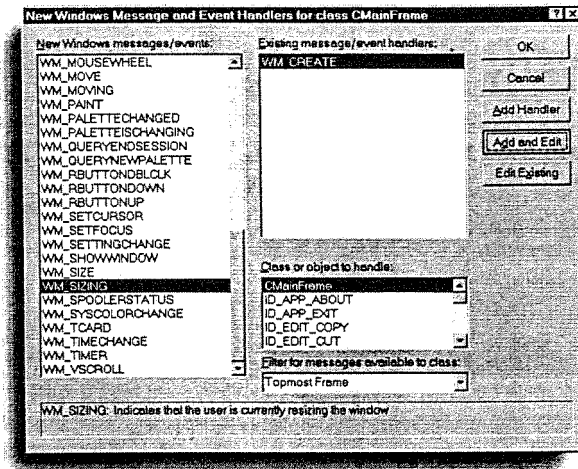
Adăugarea unei funcții de tratare a mesajului `WM_SIZING`

1. Efectuați un clic pe semnul plus alăturat elementului **SizeForm Classes** din pagina **ClassView** a secțiunii spațiului de lucru al proiectului. Astfel veți afișa lista cu clasele proiectului.

2. Fereastra cadru se ocupă de mesaje de dimensionare, așa că pentru interceptarea mesajului vom alege clasa `CMainFrame`. Efectuați un clic cu butonul drept pe clasa `CMainFrame` pentru a afișa meniul contextual și selectați opțiunea **Add Windows Message Handler**.
3. În consecință, ar trebui să fie afișată caseta de dialog **New Windows Message and Event Handlers for Class CMainFrame**.
4. Derulați mesajele din caseta cu listă **New Windows Messages/Events** până când găsiți evenimentul `WM_SIZING`.
5. Selectați acest eveniment și apoi efectuați un clic pe butonul **Add and Edit**, evidențiat în figura 18.2.

FIGURA 18.2

Adăugarea unei funcții de
tratare pentru evenimentul
`WM_SIZING`.



Acum ar trebui să vedeți codul noii funcții de tratare `OnSizing()`. Adăugați liniile prezentate în listingul 18.1 pentru a afișa pe bara de titlu informațiile legate de dimensionare.

LISTINGUL 18.1 LST19_1.CPP – Afișarea parametrilor dreptunghiului de dimensionare

```
1 void CMainFrame::OnSizing(UINT fwSide, LPRECT pRect)
2 {
3     CFrameWnd::OnSizing(fwSide, pRect);
4
5     // TODO: Add your message handler code here
6
7     // ** Cream un CRect pe baza pointerului la structura
8     CRect rcSize(pRect); — ❶
9
10    // ** Cream un sir in care vom depune un mesaj
```

```
11    CString sizeMsg;
12
13    // ** Afișam informatiile legate de dimensionare
14    // ** pe baza dreptunghiului de dimensionare
15    sizeMsg.Format("Sizing: width = %d, height = %d, — ❷
16                  rcSize.Width(), rcSize.Height());
17
18    // ** Deblocam fereastra
19    UnlockWindowUpdate(); — ❸
20
21    // ** Actualizam textul de pe bara de titlu
22    SetWindowText(sizeMsg);
23
24    // ** Blocam fereastra la loc
25    LockWindowUpdate();
26 }
```

❶ Structura `RECT` este mai ușor de folosit în urma conversiei la un obiect `CRect`.

❷ Dimensiunea atinsă curent este formatată într-un `CString`, pregătind afișarea ca titlu al ferestrei.

❸ Din cauză că Windows blochează fereastra în timpul redimensionării, va trebui să o deblocați temporar.

Această funcție de tratare va fi apelată acum la fiecare deplasare a mouse-ului din timpul redimensionării ferestrei. Funcția primește un indicator care precizează latura(-ile) ferestrei care se deplasează (`fwSide`) și un pointer la un dreptunghi care conține coordonate (`pRect`). Mai întâi trebuie să converțiți structura `RECT` la un mai comod obiect `CRect`, ca în linia 8. După aceea se construiește un șir care conține lățimea și înălțimea dreptunghiului. În fine, acest șir devine noul titlu al ferestrei prin apelul funcției `SetWindowText()` din linia 22, coordonatele fiind astfel afișate.

Observați apelurile `UnlockWindowUpdate()` și `LockWindowUpdate()` care încadrează apelul `SetWindowText()` (liniile 19 și 25). Acestea sunt necesare din cauză că Windows blochează automat desenarea în cadrul ferestrei pe durata dimensionării. În mod normal, acest lucru este benefic; în caz contrar, suprafața ferestrei ar deveni un dezastru. Aplicația ar tot încerca să deseneze în timp ce ajustați dreptunghiul de dimensionare.

Rularea exemplului `SizeForm` și pachetul Microsoft Plus!

În cazul în care ați instalat pe calculator pachetul Microsoft Plus!, ar trebui să vă asigurați că opțiunea **Show Windows Contents While Dragging** este dezactivată (nu este bifată). Această opțiune se găsește în secțiunea **Display** a panoului de control, în cadrul paginii **Plus!**. În caz contrar, fereastra nu va mai fi blocată și afișarea o va lua razna.

Dacă asamblați și rulați aplicația după efectuarea acestor modificări, lățimea și înălțimea ferestrei ar trebui să apară pe bara de titlu a ferestrei pe măsură ce o dimensionați.

Tratarea evenimentului de fixare a dimensiunii

Evenimentul de dimensionare nu este probabil la fel de util precum evenimentul de fixare a dimensiunii. Puteți să interceptați evenimentul de dimensionare pentru a manipula structura dreptunghi transmisă, dar așa ceva nu se obișnuiește. Evenimentul de fixare a dimensiunii este mai util. Acesta arată că utilizatorul a atins dimensiunea dorită a ferestrei și a eliberat butonul mouse-ului. Mesajul asociat poate fi interceptat de către fereastra cadru, așa cum ați procedat în cazul mesajului de dimensionare, dar va fi transmis mai departe ferestrei reprezentare, pentru care este probabil mai relevant. Acum adăugați funcția de tratare corespunzătoare.

Adăugarea în clasa reprezentare a unei funcții de tratare pentru evenimentul de fixare a dimensiunii

1. Selectați clasa `CSizeFormView` din pagina ClassView a secțiunii spațiului de lucru al proiectului.
2. Efectuați un clic cu butonul drept pe această clasă și apoi selectați **Add Windows Message Handler**.
3. Va fi afișată caseta de dialog New Windows Message and Event Handlers corespunzătoare clasei reprezentare.
4. Selectați evenimentul `WM_SIZE` din lista **New Windows Message/Events**.
5. Efectuați un clic pe butonul **Add and Edit** pentru a trece la adăugarea codului necesar.

Operați modificările ilustrate în listingul 18.2 pentru a afișa pe bara de titlu lățimea și înălțimea fixate.

LISTINGUL 18.2 LST19_2.CPP – Implementarea unei funcții de tratare a evenimentului de fixare a dimensiunii

```
1 void CSizeFormView::OnSize(UINT nType, int cx, int cy)
2 {
3     CFormView::OnSize(nType, cx, cy);
4
5     // TODO: Add your message handler code here
6
7     // ** Declaram un obiect sir de caractere
8     CString strTitle;
9
10    // ** Pregatim si afisam titlul ferestrei
11    strTitle.Format("Final width = %d, Height = %d", cx, cy);
12    GetDocument()->SetTitle(strTitle);
13 }
```

❶ Dimensiunile sunt formate și apoi stabilite ca titlu al documentului (acesta fiind afișat pe bara de titlu a ferestrei).

Funcția de tratare `OnSize()` prezentată în listingul 18.2 va fi apelată de fiecare dată când fereastra este redimensionată. Windows transmite prin primul parametru, `nType`, al funcției de tratare o valoare indicator. Aceasta precizează în ce fel a fost redimensionată fereastra și poate avea una dintre valorile enumerate în tabelul 18.1.

TABELUL 18.1 Indicatori pentru evenimentul de redimensionare

Indicator	Descriere
<code>SIZE_RESTORED</code>	Fereastra a fost redimensionată
<code>SIZE_MAXIMIZED</code>	Fereastra a fost maximizată
<code>SIZE_MINIMIZED</code>	Fereastra a fost minimizată

Ceilalți doi parametri transmiși funcției `OnSize()` din listingul 18.2 sunt întregi reprezentând noua lățime (`cx`) și noua înălțime (`cy`). Aceste valori pot fi formate în scopul afișării, ca în linia 8, și înscrise pe bara de titlu a ferestrei prin apelul funcției `SetTitle()`, în linia 9.

Dacă asamblați și rulați programul după adăugarea acestei funcții de tratare, veți vedea efectul mesajelor de dimensionare pe măsură ce ajustați mărimea ferestrei și, respectiv, efectul mesajului de fixare a dimensiunii în momentul în care încheiați operația de dimensionare.

Poate că vă întrebați când și de ce ați dori să fiți informat în legătură cu redimensionarea ferestrei. O situație tipică este aceea în care doriți să ajustați dimensiunile controalelor dintr-o reprezentare de tip formular. Închipuiți-vă că ați scris un mic editor de text care folosește un control de editare multi-linie. În momentul în care utilizatorul redimensionează fereastra aplicației, veți dori ca mărimea controlului de dimensionare să se modifice corespunzător. În cele ce urmează veți vedea cum puteți să adăugați un astfel de control de editare.

Adăugarea în proiectul cu reprezentare de tip formular a unui control de editare multi-linie și derulabil

1. Efectuați un clic pe pagina Resource din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus alăturat catalogului **Resources** pentru a deschide resursele proiectului.
3. Efectuați un clic pe semnul plus alăturat catalogului **Dialog** pentru a afișa machetele de dialog existente.
4. Veți vedea o casetă de dialog al cărei nume provine din cel al proiectului (`IDD_SIZEFORMFORM`). Aceasta este macheta de dialog pentru reprezentarea de tip formular.
5. Efectuați un dublu clic pe acest element și veți vedea o casetă de dialog care nu conține decât o etichetă **TODO: Place Controls Here**.
6. Ștergeți această etichetă efectuând un clic pe suprafața sa și apăsând tasta Delete.

7. Selectați controlul casetă de editare din paleta Controls și plasați-l pe suprafața machetei de dialog.
8. Acum dimensionați controlul casetă de editare astfel încât să acopere toată suprafața punctată a machetei de dialog.
9. Apăsați Alt+Enter pentru a afișa caseta de dialog Edit Properties.
10. Efectuați un clic pe pagina Styles și apoi validați opțiunile **Multiline**, **Horizontal Scroll**, **Auto HScroll**, **Vertical Scroll**, **Auto VScroll** și **want Return**.

Utilizarea opțiunilor pentru derulare

Selectarea opțiunilor pentru derulare (**Horizontal Scroll** și **Vertical Scroll**) duce la afișarea în cadrul controlului de editare a barelor de derulare orizontală și, respectiv, verticală. Opțiunile **Auto HScroll** și **Auto VScroll** vor determina controlul de editare să efectueze automat derularea la sfârșitul unei linii sau, respectiv, atunci când utilizatorul apasă tasta Enter. În cazul în care opțiunea **Auto HScroll** nu este selectată, textul conținut va fi desfășurat automat pe următoarea linie a controlului de editare.

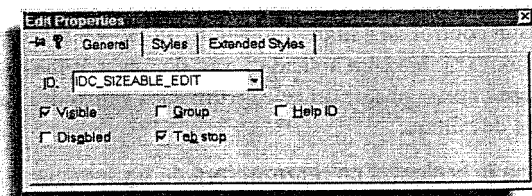
11. Selectați pagina **General** și fixați indicatorul de resursă la **IDC_SIZEABLE_EDIT**, așa cum se vede în figura 18.3.

12. Apăsați Enter pentru a închide caseta de dialog cu proprietăți.

Acum ați adăugat controlul de editare, dar nu există nici o variabilă din program care să fie asociată acestui control pentru a putea realiza o comunicație. Următoarea sarcină este să mapați peste control o variabilă cu ajutorul ClassWizard.

FIGURA 18.3

Adăugarea controlului de editare multi-linie.



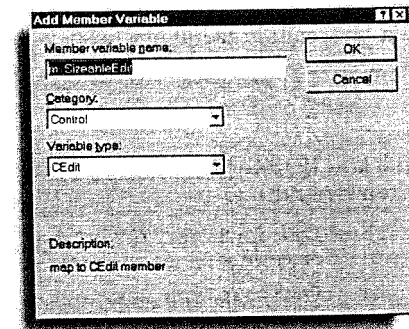
Maparea unei variabile peste un control de editare cu ajutorul ClassWizard

1. Efectuați cu grijă un clic pe marginea controlului de editare.
2. Apăsați Ctrl+W pentru a apela ClassWizard (sau efectuați un clic pe meniul **View** și selectați **ClassWizard**).
3. Va fi afișată pagina caseta de dialog ClassWizard, fiind selectată pagina **Member Variables**.
4. Selectați identificatorul de control adecvat din lista **Control IDs** (în cazul nostru, **IDC_SIZEABLE_EDIT**). Efectuați un clic pe **Add Variable** pentru a afișa caseta de dialog Add Member Variable.
5. Acum puteți să adăugați o variabilă care va reprezenta controlul de editare.
6. Selectați **Control** din caseta combinată **Category**; observați că în câmpul **Variable Type** va apărea automat **CEdit**.

7. Efectuați un clic pe caseta de editare **Member Variable Name** și introduceți numele noii variabile mapate (de pildă, **m_SizeableEdit**), așa cum se vede în figura 18.4.

FIGURA 18.4

Adăugarea unei variabile de tip **CEdit** mapată peste controlul de editare.



8. Efectuați un clic pe **OK** pentru a adăuga variabila membru în clasa reprezentare.

Acum aveți o variabilă **m_SizeableEdit** mapată peste control. Aceasta vă permite să transmiteți text de la program la control și invers. Dar această variabilă poate fi utilizată și pentru a trimite controlului mesaje de dimensionare.

Următorul lucru pe care trebuie să-l faceți este să interceptați mesajul **WM_SIZE**, astfel încât să știți când și cu cât trebuie să modificați dimensiunile controlului de editare. Există deja o funcție de tratare adecvată și o puteți accesa chiar din ClassWizard, așa cum o arată secvența de pași următoare.

Accesarea unei funcții membru/de tratare prin intermediul ClassWizard

Accesarea unei funcții membru prin intermediul paginii ClassView

O altă soluție de a accesa funcția **OnSize()** este să o găsiți în cadrul paginii **ClassView** a secțiunii spațiului de lucru al proiectului. Această funcție membru este subordonată elementului **CSizeFormView** și poate fi afișată imediat într-o fereastră de editare prin efectuarea unui dublu clic pe numele său. Ca alternativă, dacă veți efectua un clic cu butonul drept pe numele funcției veți putea să accesați definiția funcției sau declarația ei, selectând, respectiv, opțiunile **Go to Definition** sau **Go to Declaration**.

1. Selectați pagina **Message Maps** a casetei de dialog ClassWizard și asigurați-vă că în câmpul **Class Name** este selectat numele clasei în cauză (**CSizeFormView**).
2. Derulați lista **Member Functions** pentru a găsi mesajul (**ON_WM_SIZE**) sau funcția membru (**OnSize()**) dorită.
3. Efectuați un dublu clic pe elementul găsit și vă veți găsi în cadrul funcției membru a clasei selectate (în cazul nostru, **OnSize()** din clasa reprezentare formular, **CSizeFormView**).

Deocamdată nu există cod care să modifice mărimea controlului de editare odată cu redimensionarea ferestrei. Dacă asamblați și rulați aplicația, veți vedea controlul de editare afișat în fereastra reprezentării. Încercați să redimensionați fereastra și veți observa că

mărimea controlului de editare nu se modifică. Puteți să introduceți text în cadrul controlului ca într-un mini-editor, însă orice editor de text decent va ajusta fereastra de editare pe măsura dimensiunilor ferestrei aplicației.

Închideți aplicația și reveniți în cadrul funcției `OnSize()` a clasei reprezentare de tip formular (`CSizeFormView`). Adăugați la sfârșitul acestei funcții liniile de cod prezentate în listingul 18.3.

LISTINGUL 18.3 LST19_3.CPP – Redimensionarea controlului de editare odată cu redimensionarea ferestrei

```
1 void CSizeFormView::OnSize(UINT nType, int cx, int cy)
2 {
3     CFormView::OnSize(nType, cx, cy);
4
5     // TODO: Add your message handler code here
6
7     CString strTitle;
8     strTitle.Format("Final Width = %d,
9     Height = %d", cx, cy);
10
11     GetDocument()->SetTitle(strTitle);
12
13
14     // ** Verificam 'validitatea' casetei de editare
15     if (m_SizeableEdit.GetSafeHwnd())
16
17         // ** Ajustam in functie de dimensiunea ferestrei
18         m_SizeableEdit.SetWindowPos(this, 0, 0, — ❶
19                                     cx-40, cy-40,
20                                     // ** Doar redimensionam
21                                     SWP_NOMOVE+SWP_NOZORDER+SWP_SHOWWINDOW+
22                                     SWP_NOACTIVATE);
23 }
```

❶ Dimensiunile stabilite pentru controlul de editare sunt puțin mai mici decât lățimea și înălțimea ferestrei. Indicatorii specificați arată că se dorește exclusiv redimensionarea.

Trebuie să apelați funcția membru `GetSafeHwnd()` a controlului de editare pentru a vă asigura că acesta a fost inițializat, așa cum se întâmplă în linia 12 a listingului 18.3 (altfel, nu trebuie să efectuați nici o operație asupra unei ferestre care nu a fost inițializată). Funcția `SetWindowPos()` apelată în linia 15 este o funcție „știe tot, face tot” care vă permite să deplasați, să dimensionați, să activați sau să ascundeți o fereastră. În acest caz urmăriți exclusiv redimensionarea, motiv pentru care veți transmite noile dimensiuni `cx` și `cy` ale ferestrei (minus 40 de pixeli). Indicatorii combinați în cadrul ultimului parametru al funcției `SetWindowPos()` precizează operațiile care se doresc efectuate și semnificațiile celorlalți parametri. Acești indicatori sunt prezentați împreună cu explicațiile de rigoare în tabelul 18.2.

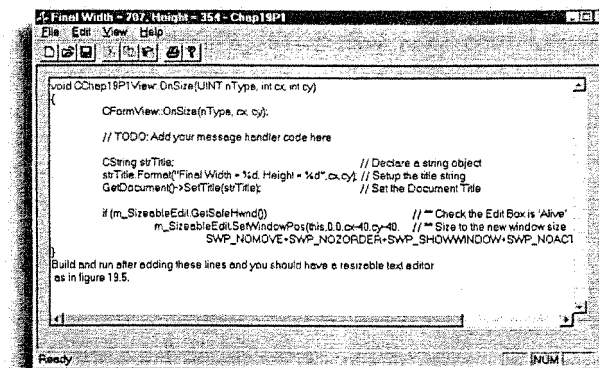
TABELUL 18.2 Indicatorii pentru `SetWindowPos()`

Indicator	Descriere
<code>SWP_NOMOVE</code>	Fereastra nu este deplasată; al doilea și al treilea parametru (<code>x</code> și <code>y</code>) se ignoră
<code>SWP_NORESIZE</code>	Fereastra nu este redimensionată; al patrulea și al cincilea parametru (<code>cx</code> și <code>cy</code>), corespunzători dimensiunii ferestrei, se ignoră
<code>SWP_NOZORDER</code>	Fereastra nu este adusă în fața sau în spatele altor ferestre (se ignoră primul parametru, <code>pWndInsertAfter</code>)
<code>SWP_NOACTIVATE</code>	Fereastra nu este activată
<code>SWP_SHOWWINDOW</code>	Fereastra este afișată

Asamblați și rulați aplicația după adăugarea acestor linii și veți obține un editor de text care poate fi redimensionat, înfățișat în figura 18.5.

FIGURA 18.5

Aplicația editor de text poate fi redimensionată.



Stabilirea de limite pentru dimensiuni

Ați putea dori să împiedicați utilizatorul să mărească sau să micșoreze fereastra în afara unor anumite limite. În acest scop puteți să interceptați în fereastra cadru mesajul `WM_GETMINMAXINFO`, prin intermediul unei funcții de tratare `OnGetMinMaxInfo()`. Acest mesaj poate fi prins în momentul în care este transmis ferestrei cadru, permițându-vă să stabiliți limite maximă și minimă pentru dimensiuni.

Adăugarea unei funcții de tratare care să controleze dimensiunile ferestrei în perspectiva unor limite maximă și minimă

Adăugarea funcției de tratare prin WizardBar

Opțiunea **Add Windows Message Handler** este accesibilă și prin intermediul săgeții de derulare din extremitatea dreaptă a barei cu instrumente WizardBar. În cazul în care folosiți această bară cu instrumente în locul paginii `ClassView`, trebuie să vă asigurați că în prima caseta combinată (cea mai din stânga) de pe această bară este înscrisă clasa `CMainFrame`, astfel încât noua funcție de tratare să fie inserată în această clasă.

1. Efectuați un clic cu butonul drept pe clasa `CMainFrame` în cadrul paginii `ClassView` a secțiunii spațiului de lucru al proiectului.
2. Selectați **Add Windows Message Handler** din meniul de context. Va fi afișată caseta de dialog **New Windows Message and Event Handlers for Class CMainFrame**.
3. Selectați mesajul `WM_GETMINMAXINFO` din lista **New Windows Messages/Events** și efectuați un clic pe butonul **Add and Edit** pentru a adăuga noua funcție de tratare.

Introduceți în noua funcție de tratare `OnGetMinMaxInfo()` codul prezentat în listingul 18.4. Efectul său va fi stabilirea dimensiunilor maximă și minimă ale ferestrei, împiedicând astfel utilizatorul să depășească anumite limite.

LISTINGUL 18.4 LST19_4.CPP – Stabilirea dimensiunilor minimă și maximă ale ferestrei cu `OnGetMinMaxInfo()`

```
1 void CMainFrame::OnGetMinMaxInfo(MINMAXINFO FAR* lpMMI)
2 {
3     // TODO: Add your message handler code here and/or c
4
5     // ** Stabilim dimensiunea minima
6     lpMMI->ptMinTrackSize = CPoint(200,200);
7
8     // ** Stabilim dimensiunea maxima
9     lpMMI->ptMaxTrackSize = CPoint(500,400);
10
11     CFrameWnd::OnGetMinMaxInfo(lpMMI);
12 }
```

Dacă asamblați și rulați aplicația după adăugarea acestor linii de cod, încercați să redimensionați fereastra. Veți observa că nu puteți să o micșorați sub 200x200 pixeli și nici nu puteți să o măriți peste 500x400 pixeli. Prin modificarea membrilor `ptMinTrackSize` (în linia 6) și `ptMaxTrackSize` (în linia 9) ai structurii `MINMAXINFO` ați stabilit dimensiunile minimă și maximă ale ferestrei.

Structura `MINMAXINFO` este definită în listingul 18.5.

LISTINGUL 18.5 LST19_5.CPP – Structura `MINMAXINFO`

```
1 typedef struct tagMINMAXINFO {
2     POINT ptReserved;
3     POINT ptMaxSize;
4     POINT ptMaxPosition;
5     POINT ptMinTrackSize;
6     POINT ptMaxTrackSize;
7 } MINMAXINFO;
```

Tipul de date `POINT`

Tipul de date `POINT` este o structură WIN32 care reține coordonatele unui punct prin intermediul membrilor `x` și `y`. S-ar putea să fiți mai familiar cu clasa `CPoint`, care încapsulează structura `POINT` și, în plus, adaugă o serie de funcții membru și operatori supradefiniți ce pot fi utili.

Există încă două elemente asupra cărora putem să intervenim aici. Unul dintre ele este membrul `ptMaxSize` din linia 3, care vă permite să fixați lățimea și înălțimea ferestrei în starea maximizată. Celălalt este membrul `ptMinSize` din linia 4, care vă permite să fixați poziția colțului stânga sus al ferestrei în starea maximizată.

Crearea unor casete de dialog dimensionabile

Puteți face foarte ușor ca o casetă de dialog să fie dimensionabilă. Secțiunea care urmează prezintă modul în care puteți modifica în acest sens caseta de dialog **About** a aplicației.

Transformarea casetei **About** într-o fereastră dimensionabilă

1. Efectuați un clic pe pagina `ResourceView` din secțiunea spațiului de lucru al proiectului.
2. Afișați resursele de tip dialog efectuând clic pe semnele plus alăturate resurselor proiectului (**SizeForm Resources**) și elementului **Dialog**.
3. Efectuați un dublu clic pe caseta de dialog `IDD_ABOUTBOX` pentru a o edita.
4. Apăsați **Alt+Enter** pentru a accesa proprietățile acestei casete de dialog.
5. Derulați caseta combinată **Border** din pagina **Styles** și selectați opțiunea **Resizing**.
6. Asamblați și rulați aplicația.
7. Efectuați un clic pe meniul **Help** al aplicației și selectați opțiunea **About** corespunzătoare (de exemplu, **About SizeForm**).

Acum ar trebui să puteți dimensiona caseta de dialog asemeni oricărei alte ferestre. Dacă doriți, puteți să tratați mesajele `OnSize()` exact așa cum ați procedat mai devreme, în exemplul de editor de text.

Derularea ferestrelor

Nu este întotdeauna posibil să afișați în întregime o reprezentare sau un formular la dimensiunile curente ale ferestrei. Uneori ați putea dori chiar să desenați pe o suprafață mai mare decât ecranul, permițând utilizatorului să deruleze această suprafață. MFC vă oferă o clasă reprezentare foarte utilă care se numește `CScrollView` și face posibile toate aceste ipostaze. Probabil că ați utilizat multe aplicații care conțin bare de derulare și vă permit să derulați zona vizibilă a unei ferestre. Puteți să creați cu `AppWizard` o aplicație care să folosească clasa `CScrollView` pentru a oferi o astfel de zonă derulabilă.

Pașii care trebuie urmați sunt aproape identici cu cei pentru crearea aplicației cu o reprezentare de tip formular, prezentați în secvența „Crearea unei aplicații SDI cu o

reprezentare de tip formular”. În ultima casetă de dialog, însă, nu veți mai selecta CFormView în caseta combinată **Base Class**, ci CScrollView.

Exemplele următoare prezintă aspecte ale unei aplicații ce conține o reprezentare de tip CScrollView, aplicație numită PanSDI.

Stabilirea dimensiunilor de derulare

Dacă asamblați și rulați un proiect SDI cu o reprezentare CScrollView, aspectul său nu va diferi cu nimic de o aplicație SDI a cărei reprezentare este de tip CView. Motivul este faptul că dimensiunea totală implicită pentru suprafața derulabilă a acestei aplicații este de 100x100 pixeli. Această dimensiune implicită poate fi modificată astfel încât suprafața reprezentării să o depășească pe cea a ecranului, modificând în acest sens implementarea implicită OnInitialUpdate(), așa cum se arată în continuare.

Editarea funcției OnInitialUpdate() a clasei reprezentare în scopul modificării dimensiunii suprafeței de derulare

1. Selectați pagina ClassView din secțiunea spațiului de lucru al proiectului.

Funcția OnInitialUpdate()

OnInitialUpdate() este un loc potrivit pentru efectuarea o singură dată a unor inițializări ale reprezentării, deoarece această funcție este apelată doar o dată, la inițializarea ferestrei reprezentării. Este similară cu funcția OnInitDialog() a unei casete de dialog. Trebuie, însă, să aveți grijă la referirea obiectului document al aplicației în cadrul funcției OnInitialUpdate(), deoarece uneori reprezentarea este inițializată înaintea documentului. Aceasta ar putea duce la blocări aleatoare ale aplicației la momentul lansării.

2. Deschideți clasele proiectului (de exemplu, PanSDI) efectuând un clic pe simbolul plus.
3. Deschideți clasa de tip CScrollView (de pildă, CPanSDIView) efectuând un clic pe semnul plus alăturat acesteia. În partea inferioară a listei funcțiilor membru veți găsi funcția OnInitialUpdate().
4. Efectuați un dublu clic pe această funcție și veți vedea implementarea implicită, prezentată în listingul 18.6.
5. Dimensiunile implicite ale suprafeței de derulare pot fi ajustate prin modificarea valorilor sizeTotal.cx și sizeTotal.cy.

LISTINGUL 18.6 LST19_6.CPP – Implementarea standard a funcției OnInitialUpdate() pentru o reprezentare derulabilă

```
1 void CPanSDIView::OnInitialUpdate()
2 {
3     CScrollView::OnInitialUpdate();
4     CSize sizeTotal;
5
6     // TODO: calculate the total size of this view
```

```
7     sizeTotal.cx = sizeTotal.cy = 100;
8     SetScrollSizes(MM_TEXT, sizeTotal);
9 }
```

- ❶ În mod implicit, dimensiunea totală a suprafeței de derulare este de doar 100x100 pixeli, de regulă prea mică pentru a mai putea permite o derulare.

Funcția OnInitialUpdate() (prezentată în listingul 18.6) este apelată o singură dată, chiar înainte ca reprezentarea să fie afișată pentru prima dată, dar după ce a fost atașată la document. Această funcție poate fi utilizată pentru implementarea unor operații de inițializare a reprezentării care se vor efectua o singură dată, înainte de afișare.

Linia 7 a listingului 18.6 inițializează un obiect CSize la 100x100; acest obiect va fi transmis apoi funcției SetScrollSizes(). Aceasta este funcția care stabilește dimensiunea totală a reprezentării în raport cu un anumit mod de mapare. Modul MM_TEXT face ca dimensiunea să fie considerată în pixeli. (Modurile de mapare sunt discutate amănunțit în capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”). Așadar, suprafața reprezentării este fixată la 100x100 pixeli; în mod normal, dimensiunile ferestrei sunt mai mari decât atât, așa că nu sunt afișate bare de derulare.

S-ar putea să aveți nevoie de o suprafață de reprezentare foarte mare, poate chiar mai mare decât întreg ecranul. În acest caz ați putea să înlocuiți suprafața de 100x100 pixeli cu una de 2000x2000 pixeli, înlocuind 100 cu 2000 ca în listingul 18.7. Aceste modificări vor mări suprafața de reprezentare peste dimensiunea ecranului (dacă nu cumva aveți un monitor cu o rezoluție extrem de mare!).

LISTINGUL 18.7 LST19_7.CPP – Stabilirea unei suprafețe de reprezentare mai mare decât ecranul

```
1 void CPanSDIView::OnInitialUpdate()
2 {
3     CScrollView::OnInitialUpdate();
4     CSize sizeTotal;
5
6     // TODO: calculate the total size of this view
7     sizeTotal.cx = sizeTotal.cy = 2000; // ** Noua dimensiune
8     SetScrollSizes(MM_TEXT, sizeTotal);
9 }
```

- ❶ 2000x2000 pixeli este o dimensiune mai mare decât cea a ecranului, așa că fereastra reprezentării va trebui derulată, prezentând la un moment dat utilizatorului o porțiune din întreaga reprezentare.

Puteți să desenați ceva mare în cadrul reprezentării pentru a vedea efectele derulării. De exemplu, adăugați noile linii din listingul 18.8 pentru a trasa o elipsă. Funcția OnDraw() poate fi accesată efectuând un dublu clic pe numele său din cadrul paginii ClassView a secțiunii spațiului de lucru al proiectului.

LISTINGUL 18.8 LST19_8.CPP – Desenarea unei imagini mai mari decât ecranul în cadrul unei reprezentări derulabile

```

1 void CPanSDIView::OnDraw(CDC* pDC)
2 {
3     CPanSDIDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7
8     // ** Selectam o pensula gri
9     CBrush* pOldBrush =
10     (CBrush*)pDC->SelectStockObject(LTGRAY_BRUSH);
11
12     // ** Cream un CRect
13     CRect rcTotal(CPoint(0,0),GetTotalSize());
14
15     // ** Desenam elipsa
16     pDC->Ellipse(rcTotal); ❶
17
18     // ** Selectam la loc pensula originala
19     pDC->SelectObject(pOldBrush);
20 }

```

❶ Elipsa desenată va fi mai mare decât fereastra și, prin urmare, va fi decupată la zona vizibilă a ferestrei.

Observați în listingul 18.8 că funcția `GetTotalSize()` a reprezentării derulabile este utilizată (în linia 13) pentru a determina dimensiunea totală a suprafeței reprezentării, stabilită de apelul `SetScrollSizes()` din `OnInitialUpdate()`. Pe baza acestei informații este construit un obiect `CRect` care va fi folosit la desenarea unei elipse de 2000x2000 pixeli. Pentru trasarea elipsei este utilizată (în linia 10) o pensulă gri din stoc (`LTGRAY_BRUSH`).

Dacă ați asambla și rula aplicația după efectuarea acestor modificări, veți vedea o fereastră care conține o parte dintr-un cerc uriaș (așa cum o arată figura 18.6). Prin acționarea barelor de derulare veți putea să vedeți și alte porțiuni ale cercului. Ați putea încerca să redimensionați fereastra pentru a observa cum se modifică barele de derulare astfel încât să reprezinte ceea ce este afișat.

Modificarea incrementelor de derulare pentru linie și pagină

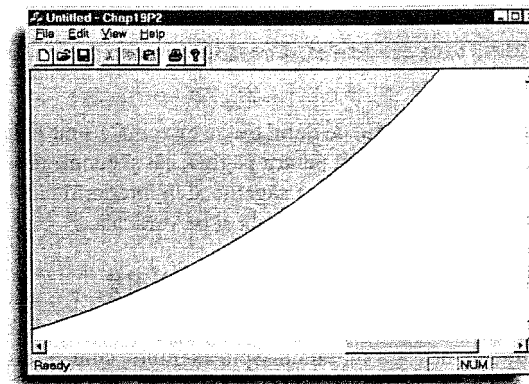
Dacă efectuați clic pe săgețile aflate la capetele barelor de derulare, veți vedea că imaginea este derulată cu puțin; această valoare se numește *increment de linie*. Dacă efectuați un clic pe spațiul dintre o săgeată a barei de derulare și caseta de derulare a acesteia, veți vedea că imaginea este derulată mai mult. Această valoare reprezintă *incrementul de pagină*. Numele sunt legate evident de aplicațiile care operează cu text, așa cum sunt procesoarele de text, dar sunt la fel de valabile pentru orice este afișat în cadrul reprezentării.

Incrementele de derulare pentru linie și pagină

În mod normal, incrementele de derulare pentru linie și pagină se exprimă în pixeli, împreună cu indicatorul de mapare `MM_TEXT`. Aceste incremente pot fi precizate, însă, în orice alt mod de mapare, astfel încât să corespundă ipostazei vizate. Spre exemplu, ați putea configura barele de derulare astfel încât derularea să se producă cu 1cm pentru fiecare linie și cu 10cm pentru fiecare pagină.

FIGURA 18.6

Afișarea unor obiecte mai mari decât fereastra cu ajutorul unei reprezentări derulabile.



Distanțele cu care se efectuează derularea la acționarea elementelor corespunzătoare ale barelor de derulare se pot modifica prin transmiterea unor parametri suplimentari în apelul `SetScrollSizes()`. Dacă omiteți acești parametri, incrementele de linie și de pagină primesc valori implicite. Pentru a modifica aceste incremente, atât pentru bara de derulare orizontală, cât și pentru cea verticală, trebuie să transmiteți obiecte `CSize` corespunzătoare. Membrii obiectului `CSize` sunt `cx` și `cy`. `cx` stabilește incrementul de linie sau de pagină pentru bara de derulare orizontală; `cy` stabilește incrementul de linie sau de pagină pentru bara de derulare verticală.

Funcția `SetScrollSizes()` poate fi apelată înainte de prima afișare a reprezentării, în cadrul funcției `OnInitialUpdate()`. Pentru a edita funcția membru `OnInitialUpdate()` a clasei reprezentare puteți să efectuați un dublu clic pe numele său din cadrul paginii `ClassView` a secțiunii spațiului de lucru al proiectului.

Modificarea incrementelor de linie și de pagină se face prin transmiterea unor obiecte `CSize` corespunzătoare în apelul funcției `SetScrollSizes()` (vedeți listingul 18.9).

LISTINGUL 18.9 LST19_9.CPP – Fixarea incrementelor de pagină și de linie cu `SetScrollSizes()`

```

1 void CPanSDIView::OnInitialUpdate()
2 {
3     CScrollView::OnInitialUpdate();
4     CSize sizeTotal;

```

(continuare)

LISTINGUL 18.9 Continuare

```

5 // TODO: calculate the total size of this view
6 sizeTotal.cx = sizeTotal.cy = 2000; // Noua dimensiune
7 SetScrollSizes(MM_TEXT, sizeTotal,
8     CSize(200,10), // ** Incrementele de pagina (X,Y) — ❶
9     CSize(20,1)); // ** Incrementele de linie (X,Y)
10 }

```

❶ SetScrollSize() poate, de asemenea, să modifice incrementele de pagină și de linie pentru fiecare dintre barele de derulare.

Observați că incrementele de derulare pe orizontală sunt mult mai mari decât incrementele de derulare pe verticală, atât pentru pagină, cât și pentru linie. Dacă asamblați și rulați aplicația și apoi efectuați clic pe săgețile și în interiorul barelor de derulare, veți sesiza diferența între derularea pe orizontală și pe verticală.

Utilizarea poziției curente de derulare

Ați putea dori să aflați ce porțiune din reprezentare este vizibilă curent și care este poziția acesteia față de întreaga suprafață. Spre exemplu, cum ați proceda în cazul în care doriți să afișați o cruciuliță în centrul ferestrei? Este posibil să folosiți funcția `GetClientRect()` și membrul `CenterPoint()` al obiectului `CRect`, ca mai înainte? Ei bine, aceste funcții v-ar furniza centrul ferestrei în cazul în care colțul din stânga sus al acesteia ar coincide cu cel al reprezentării. Dar dacă barele de derulare au fost acționate în sensul afișării unei zone din centrul reprezentării, centrul dreptunghiului client ar fi undeva în partea din stânga sus a reprezentării și acolo ar fi trasată și cruciulița. Funcția `GetScrollPosition()` a unei reprezentări derulabile vă arată unde se află colțul stânga sus al porțiunii vizibile în raport cu întreaga reprezentare. Dacă adunați această valoare cu rezultatul întors de `CenterPoint()`, veți reuși să desenați cruciulița în mijlocul zonei vizibile.

Determinarea poziției de derulare în cadrul dispozitivului

Funcția `GetScrollPosition()` întoarce poziția de derulare curentă exprimată în unități logice. Prin urmare, aceste unități depind de modul de mapare stabilit. Dacă aveți nevoie de poziția de derulare exprimată în unități dispozitiv (pixeli), va trebui să apelați la funcția `GetDeviceScrollPosition()`. Cele două funcții întorc același rezultat atunci când modul de mapare este `MM_TEXT`.

Puteți să adăugați după apelul `Ellipse()` din funcția `OnDraw()` liniile de cod prezentate în listingul 18.10, trasând astfel o cruciuliță în centrul regiunii afișate.

LISTINGUL 18.10 LST19_10.CPP – Trasarea unei cruciulițe în centrul ferestrei cu ajutorul funcției `GetScrollPosition()`

```

1 // Cream un CRect
2 CRect rcTotal(CPoint(0,0),GetTotalSize());
3

```

```

4 // Desenam elipsa
5 pDC->Ellipse(rcTotal);
6
7 // ** Determinam dreptunghiul client
8 CRect rcClient;
9 GetClientRect(&rcClient);
10
11 // ** Luam în calcul poziția de derulare
12 rcClient += GetScrollPosition(); — ❶
13
14 // ** Determinam centrul
15 CPoint ptCenter = rcClient.CenterPoint();
16
17 // ** Linia din stanga-sus în dreapta-jos
18 pDC->MoveTo(ptCenter + CPoint(-30,-30));
19 pDC->LineTo(ptCenter + CPoint(+30,+30));
20
21 // ** Linia din dreapta-sus în stang-jos
22 pDC->MoveTo(ptCenter + CPoint(+30,-30));
23 pDC->LineTo(ptCenter + CPoint(-30,+30));
24
25 // Selectam la loc pensula originală
26 pDC->SelectObject(pOldBrush);
27 }

```

❶ Dreptunghiul client conține în mod adecvat lățimea și înălțimea regiunii vizibile. Dar pentru a desena corect pe suprafața reprezentării trebuie să adunați deplasamentele curente de derulare, furnizate de funcția `GetScrollPosition()`.

În afară de adunarea la `rcClient` a rezultatului întors de `GetScrollPosition()`, în linia 12, funcția de desenare nu are alte aspecte deosebite. Punctul central, `rcClient.CenterPoint()` (determinat în linia 15), este folosit în apelurile funcțiilor `MoveTo()` și `LineTo()` pentru desenarea cruciuliței, așa cum se vede în liniile de la 18 la 23.

Totul pare în regulă, dar există o problemă ascunsă care va împiedica aplicația să funcționeze corect. Dacă asamblați și rulați programul după efectuarea acestor modificări, cruciulița va fi afișată și totul arată bine. Dar în momentul în care încercați să derulați reprezentarea, cruciulița dispare.

Aceasta se întâmplă din cauza operației de *decupare*. Windows încearcă să reducă la minimum efortul depus astfel încât să accelereze cât mai mult execuția. În momentul în care derulați suprafața reprezentării, Windows se gândește „Bun, nu trebuie să redesenez decât noua porțiune care a apărut în fereastră”. Prin urmare, se redesenează doar banda care a apărut de-a lungul unei laturi a ferestrei, iar restul este pur și simplu mutat. Totul ar merge bine dacă nu ați desena decât cercul fixat, dar cruciulița se deplasează. Prin urmare, trebuie să informați explicit Windows că ideile sale privind zona invalidă din fereastră sunt greșite. În acest scop există funcții care determină Windows să invalideze părți dintr-o

fereastră sau chiar întreaga suprafață a acesteia. `Invalidate()` este o funcție utilă care determină Windows să ignore orice reguli de decupare și să redeseneze întreaga suprafață.

Invalidarea unui dreptunghi sau a unei regiuni anume

Puteți să invalidați un anumit dreptunghi, apelând funcția `InvalidateRect()` și transmitând ca prim parametru dreptunghiul respectiv. Sau puteți să utilizați funcția `InvalidateRgn()` pentru a invalida regiuni cu forme mai sofisticate, transmitând ca prim parametru un obiect `CRgn`. Aceste funcții vor actualiza regulile de decupare, iar Windows va redesea zonele pe care le indicați.

Așadar, trebuie să apelați funcția `Invalidate()`, dar unde? V-ați putea gândi la funcția `OnDraw()`, dar ar ieși un dezastru. Windows se bazează pe `OnDraw()` pentru redesenarea regiunii invalidate; dacă apelați `Invalidate()` va încerca să deseneze totul din nou și din nou și din nou...

Soluția este să aflați ce anume cauzează redesenarea și să solicitați acolo redesenarea întregii ferestre. Dar redesenarea se efectuează în urma deplasării barelor de derulare. Prin urmare, va trebui să interceptați mesajele de derulare și să apelați `Invalidate()` cu această ocazie.

Tratarea mesajelor de derulare

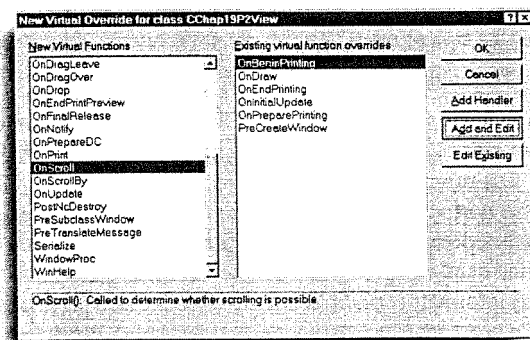
Funcția virtuală `OnScroll()` este apelată la fiecare utilizare a barelor de derulare. Această funcție poate fi supradefinită, adăugând astfel cod propriu care să răspundă la deplasările barelor de derulare.

Supradefinirea funcției `OnScroll()`

1. Efectuați un clic cu butonul drept pe clasa reprezentare derivată din `CScrollView` (de exemplu, `CPanSDView`) în cadrul paginii `ClassView` a secțiunii spațiului de lucru al proiectului. Va fi afișat un meniu contextual.
2. Selectați din acest meniu opțiunea **Add Virtual Function**. Este afișată caseta de dialog `New Virtual Override for Class CPanSDView`.
3. Derulați lista **New Virtual Functions** până când întâlniți funcția `OnScroll()`, așa cum se vede în figura 18.7.

FIGURA 18.7

Supradefinirea funcției virtuale `OnScroll` prin intermediul casetei de dialog `New Virtual Override`.



4. Efectuați un clic pe `OnScroll()` și apoi efectuați clic pe butonul **Add and Edit** pentru a adăuga supradefiniția funcției virtuale.
5. Adăugați codul care implementează funcția `OnScroll()` (vedeți apelul funcției `Invalidate()` care urmează după comentariile `// TODO:` din listingul 18.11).

LISTINGUL 18.11 LST19_11.CPP – Fortarea prin apelul `Invalidate()` a unei redesenări integrale la derularea reprezentării

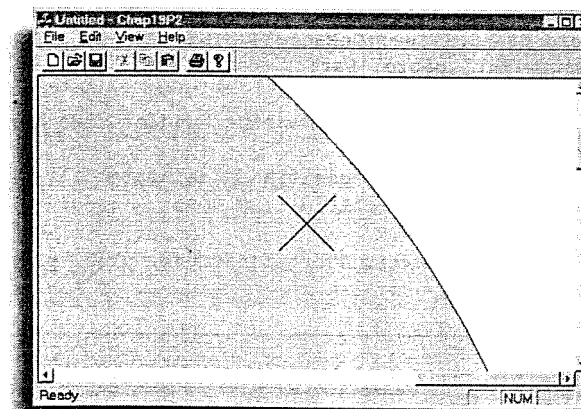
```
1 BOOL CPanSDView::OnScroll(UINT nScrollCode, UINT nPos,
  ↳BOOL bDoScroll)
2 {
3
4
5     // ** Invalidam întreaga fereastră
6     Invalidate();
7
8     return CScrollView::OnScroll(nScrollCode,
9                                   nPos, bDoScroll);
10 }
```

Cu aceasta sunt efectuate toate modificările necesare pentru desenarea cruciuliței. Apelul `Invalidate()` din linia 6 informează Windows că întreaga zonă client este invalidă și trebuie redesenată. Ceea ce desenați în funcția `OnDraw()` nu va mai fi decupat, iar cruciulița va fi afișată corespunzător după derulare.

Asamblați și rulați programul după adăugarea liniei de mai sus; acum veți putea să derulați cercul, iar cruciulița va fi afișată tot timpul în centrul ferestrei, așa cum se vede în figura 18.8.

FIGURA 18.8

Trasarea cruciuliței în urma derulării.



Puteți, de asemenea, să interceptați mesajele distincte generate de derulările pe orizontală și pe verticală, adăugând codul necesar sau prelucrând valorile curente înainte ca acestea să fie utilizate la actualizarea ferestrei și a reprezentării. În acest scop trebuie să răspundeți la mesajele `WM_HSCROLL` și `WM_VSCROLL` prin intermediul funcțiilor de tratare `OnHScroll()` și, respectiv, `OnVScroll()`.

Adăugarea unei funcții de tratare `OnHScroll()` care să răspundă la mesajul `WM_HSCROLL`

1. În pagina `ClassView` din secțiunea spațiului de lucru al proiectului, efectuați un clic cu butonul drept pe clasa reprezentare derivată din `CScrollView` (în cazul de față, `CPanSDIView`). Va apărea din nou meniul de context.
2. Selectați opțiunea **Add Windows Message Handler**. Va fi afișată caseta de dialog `New Windows Message and Event Handlers for Class CPanSDIView`.
3. Selectați mesajul `WM_HSCROLL` din lista **New Windows Messages/Events**.
4. Efectuați un clic pe butonul **Add and Edit** pentru a adăuga o funcție care să răspundă la mesajul de derulare pe orizontală.

Acum ar trebui să vedeți noua funcție de tratare `OnHScroll()` corespunzătoare mesajului `WM_HSCROLL`. Acest mesaj este trimis reprezentării la fiecare deplasare a barei de derulare orizontale. Există și o funcție de tratare `OnVScroll()` corespunzătoare mesajului `WM_VSCROLL`, mesaj trimis la deplasarea barei de derulare verticale.

Linia 6 din funcția de tratare `OnHScroll()`, prezentată în listingul 18.12, inversează poziția furnizată de bara de derulare înainte de a fi trimisă mai departe clasei reprezentare de bază.

LISTINGUL 18.12 LST19_12.CPP – Inversarea poziției barei de derulare în cadrul mesajului primit de `OnHScroll()`

```
1 void CPanSDIView::OnHScroll(UINT nSBCode, UINT nPos,
   CScrollBar* pScrollBar)
2 {
3     // TODO: Add your message handler code here
4
5     // ** Inversam pozitia de derulare
6     nPos = GetScrollLimit(SB_HORZ) - nPos;
7
8     CScrollView::OnHScroll(nSBCode, nPos, pScrollBar);
9 }
10
```

Funcția `GetScrollLimit()` este apelată în linia 6 cu scopul de a determina limitele barei de derulare, fiind transmis indicatorul `SB_HORZ` care specifică bara de derulare orizontală. Puteți să scădeți poziția curentă a barei de derulare, `nPos`, din valoarea limită a acesteia, inversând astfel acțiunea barei de derulare orizontale. Poziția `nPos` modificată este

transmisă apoi implementării din clasa de bază a funcției `OnHScroll()`, în linia 8, acolo unde mesajul de derulare este tratat ca de obicei. Numai că bara de derulare va acționa acum pe dos!

Asamblați și rulați programul după operarea acestor modificări și remarcați comportamentul ciudat al barei de derulare orizontale. Bara de derulare verticală funcționează normal (dar ați putea să modificați și comportamentul acesteia, interceptând într-un mod similar mesajul `WM_VSCROLL`).

Utilizarea reprezentărilor de tip listă, arbore, editare formatată și HTML

Întreținerea unei liste de date cu ajutorul unei reprezentări
de tip listă

Organizarea datelor ierarhice cu ajutorul unei reprezentări
de tip arbore

Implementarea unor facilități sofisticate de editare a textului
prin intermediul reprezentării de tip editare formatată

Dotarea aplicației cu facilități de navigare pe Web

Ce sunt reprezentările de tip listă, arbore și editare formatată?

Reprezentările de tip *listă*, *arbore* și *editare formatată* se bazează toate pe controale care pot fi utilizate în casetele de dialog. În linii mari, acestea funcționează prin încapsularea în cadrul reprezentării a clasei controlului corespunzător. Chiar și așa, aceste reprezentări pot constitui elemente de bază foarte puternice ale unei aplicații.

Crearea și utilizarea unei reprezentări de tip listă

Multe aplicații au în centrul activității o listă de elemente. Un exemplu evident este Windows Explorer, a cărui secțiune din dreapta este o reprezentare de tip listă. Reprezentările de tip listă vă oferă un bun control asupra modului de afișare a datelor aflate sub forma unei liste. Elementele pot fi afișate ca pictograme mari sau mici sau cu coloane care înfățișează diferite detalii. Este posibilă sortarea automată a datelor și realizarea de selecții singulare sau multiple. Toată această funcționalitate este deja implementată, permițându-vă să vă concentrați asupra aplicației și scutindu-vă de neplăcuta sarcină de a scrie codul pentru afișarea elementelor din listă.

VEDEȚI...

- Pentru utilizarea unui control listă într-o casetă de dialog, vedeți pagina 113.
- Pentru utilizarea simultană a mai multor reprezentări, așa cum există în Windows Explorer, vedeți pagina 458.
- Pentru modificarea fontului implicit cu ajutorul funcției `SetFont()`, vedeți pagina 393.

Crearea unei aplicații cu reprezentare de tip listă prin intermediul AppWizard

Puteți utiliza AppWizard pentru a genera automat o aplicație SDI în care reprezentarea standard să fie înlocuită de o reprezentare de tip listă. Haideți să creăm o astfel de aplicație care să servească drept bază de studiu pentru reprezentările de tip listă.

Crearea unei infrastructuri SDI cu reprezentare de tip listă

1. Efectuați un clic pe meniul **File** și selectați **New**.
2. Selectați pagina **Projects**; din lista cu tipurile de proiecte alegeți **MFC AppWizard (exe)**.
3. Efectuați un clic pe caseta **Project Name** și introduceți numele proiectului: `Listv`.
4. Efectuați un clic pe **OK**. Va fi afișată caseta de dialog MFC AppWizard – Step 1.
5. Selectați **Single Document** și apoi apăsați repetat pe butonul **Next** până când ajungeți la caseta de dialog care afișează un steag caroiat.
6. Efectuați un clic pe clasa `CListView` din lista claselor create de AppWizard.
7. Caseta combinată **Base Class** va fi activată. Efectuați un clic pe săgeata de derulare pentru a deschide lista cu clasele de bază posibile pentru clasa reprezentare.

8. Selectați `CListView` ca și clasă de bază pentru reprezentare.
9. Efectuați un clic pe **Finish**.
10. Efectuați un clic pe butonul **OK** din caseta de dialog New Project Information, iar AppWizard va crea noul proiect și fișierele sale sursă.

În acest moment aveți la dispoziție un schelet de aplicație SDI a cărei clasă reprezentare este derivată din `CListView`. Puteți să asamblați și să rulați aplicația acum, dar pentru că în controlul listă nu a fost înscris nici un element, nu veți vedea nimic în plus față de o aplicație SDI cu reprezentare de tip `CView`. Pentru a afișa ceva trebuie să inserați elemente în cadrul listei.

Inserarea de elemente

O aplicație reală își păstrează datele independent de reprezentare, în cadrul clasei document (sau într-o altă clasă proprie). Reprezentarea ar trebui utilizată în principal pentru operațiile care țin de prezentarea informațiilor. Așadar, locul adecvat pentru a păstra o listă de elemente este clasa `CListVDoc`, derivată din `CDocument`. Vom alcătui ca date o listă de șiruri; acestea vor fi nume de elemente chimice. MFC pune la dispoziție o clasă `CStringList` potrivită tocmai în acest scop. Adăugarea în document a datelor se poate face prin intermediul pașilor următori.

Clase colecție

Clasa `CStringList` oferită de MFC este un exemplu de ceea ce se numește clasă colecție. O clasă colecție gestionează o mulțime de date având un anumit tip. În cazul clasei `CStringList`, este evident vorba de o listă de obiecte `CString`.

Inserarea unei variabile membru prin *ClassView*

1. Selectați pagina **ClassView** din secțiunea spațiului de lucru al proiectului.
2. Deschideți clasele proiectului `ListV` pentru a putea vedea clasa `CListVDoc`.
3. Efectuați un clic cu butonul drept pe această clasă pentru a apela meniul de context.
4. Selectați opțiunea **Add Member Variable** pentru a deschide caseta de dialog Add Member Variable.
5. Introduceți `CStringList` în câmpul **Variable Type**; apoi apăsați tasta Tab pentru a accesa următorul câmp.
6. Introduceți `m_listElements` în caseta **Variable Declaration**.
7. Selectați opțiunea **Private Access** și efectuați un clic pe **OK** pentru a adăuga noua variabilă membru de tip listă de șiruri.

Acum aveți un obiect listă de șiruri încapsulat în clasa document. V-am indicat să optați pentru un acces de tip privat la variabilă pentru ca aceasta să poată fi modificată direct numai de către funcțiile membru ale clasei document. Astfel se previn accidentele determinate de modificarea inadecvată a acestei variabile listă de către funcții membru ale altor clase. Firește, este puțin probabil ca așa ceva să se întâmple într-un astfel de exemplu

mic, dar într-o aplicație mai mare veți lua asemenea măsuri pentru a permite accesul exclusiv funcțiilor sigure, membre ale documentului. Un acces de tip public ar însemna că variabila poate fi accesată de orice altă clasă. Modul protejat seamănă cu modul privat, cu deosebirea că clasele derivate nu au acces la variabilele membru private, dar pot accesa membrii protejați.

Întreaga problemă ține de *proiectarea orientată pe obiect* și nu vreau să divaghez prea mult, dar aspectul devine relevant în ceea ce privește legătura cu controlul listă încapsulat în clasa `CListView`. Așa cum este variabila noastră membru, ea nu poate fi accesată din exteriorul documentului. Aceasta va fi o problemă acum, pentru că trebuie să înscriseți în reprezentarea de tip listă conținutul listei de șiruri din cadrul documentului. Soluția este oferirea unei funcții de acces. Sarcina unei funcții de acces este să ofere un acces sigur la o variabilă membru privată. Dacă știți că singura modalitate de acces este o anumită funcție, puteți să urmăriți utilizarea funcției respective pentru a determina ce nu merge bine.

Prevenirea erorilor cu ajutorul funcțiilor de acces și al membrilor privați

Rețineți că atunci când scrieți programe de calculator, dacă ceva poate să meargă prost, va merge – mai ales atunci când prezentați noul program șefului. Protejarea variabilelor membru și utilizarea funcțiilor de acces ajută la prevenirea erorilor.

Vom adăuga în document o astfel de funcție membru care să permită accesul la lista de șiruri. Urmăți pașii de mai jos pentru adăugarea unei funcții de acces care să furnizeze lista de șiruri `m_listElements`.

Inserarea unei funcții membru prin *ClassView*

1. Efectuați un clic cu butonul drept pe clasa `CListVDoc` pentru a apela meniul contextual al clasei.
2. Selectați opțiunea **Add Member Function** a meniului pentru a afișa caseta de dialog Add Member Function.
3. Introduceți `const CStringList&` în câmpul **Function Type** și apoi apăsați tasta Tab pentru a accesa câmpul următor.
4. Introduceți `GetElements()` în câmpul **Function Declaration**.
5. Efectuați un clic pe **OK** pentru a adăuga noua funcție membru `GetElements()`.

În momentul în care efectuați clic pe **OK** veți vedea noua funcție `GetElements()`. Adăugați linia următoare pentru a întoarce o referință la lista de elemente:

```
return m_listElements;
```

Pentru că funcția întoarce o valoare de tipul `const CStringList&`, orice funcție care va accesa lista de șiruri din afara documentului va putea exclusiv să o citească. Accesul exclusiv la citire se datorează cuvântului cheie `const`. Aceasta înseamnă că clasa reprezentare poate să acceseze lista, dar nu poate să o modifice. Totul este în regulă, pentru că vreți ca lista să nu poată fi modificată decât în cadrul documentului.

Aveți nevoie de niște date pentru a testa reprezentarea de tip listă, așa că în constructorul clasei document veți adăuga câteva elemente în obiectul listă de șiruri. Pentru a edita constructorul clasei document, efectuați un dublu clic pe funcția membru `CListVDoc` din clasa `CListVDoc` (numele funcției constructor este identic cu numele clasei). Adăugați elementele de test, așa cum o arată listingul 19.1.

LISTINGUL 19.1 LST20_1.CPP – Adăugarea de elemente în cadrul funcției constructor `CListVDoc` a documentului

```
1 CListVDoc::CListVDoc()
2 {
3     // TODO: add one-time constructor code here
4     // ** Adaugam elemente in lista de siruri
5     m_listElements.AddTail("Carbon");
6     m_listElements.AddTail("Uranium");
7     m_listElements.AddTail("Gold");
8     m_listElements.AddTail("Osmium");
9     m_listElements.AddTail("Oxygen");
10    m_listElements.AddTail("Lead");
11 }
```

În listingul 19.1, liniile de la 5 la 10 adaugă diferite elemente în lista `m_listElements` transmițând șiruri de caractere funcției membru `AddTail()` a obiectului `CStringList`, funcție care adaugă șirul specificat la coada listei existente.

Următorul lucru pe care trebuie să-l faceți este să încărcați aceste date de test în reprezentarea de tip listă la prima afișare a acesteia. Singura posibilitate de accesare a listei de șiruri din cadrul reprezentării este funcția `GetElements()`, așa că va trebui să o utilizați în metoda `OnInitialUpdate()` a clasei reprezentare. `OnInitialUpdate()` este apelată o singură dată, la prima afișare a reprezentării. Pentru a accesa această funcție puteți să utilizați pagina `ClassView` din secțiunea spațiului de lucru al proiectului, efectuând un dublu clic pe numele său aflat în subordinea clasei `CListVView`. Adăugați codul prezentat în listingul 19.2 pentru a efectua încărcarea elementelor din lista membru a documentului în cadrul reprezentării de tip listă.

LISTINGUL 19.2 LST20_2.CPP – Încărcarea elementelor într-o reprezentare de tip listă cu ajutorul funcției `InsertItem()`

```
1 void CListVView::OnInitialUpdate()
2 {
3     CListView::OnInitialUpdate();
4
5     // TODO: You may populate your ListView with items
6     // its list control through GetListCtrl().
7
8     // ** Obținem un pointer la document
```

```
9     CListVDoc* pDoc = GetDocument(); — ❶
10    // ** ne asiguram ca documentul este valid
11    ASSERT_VALID(pDoc);
12
13    // ** Determinam capul listei de siruri
14    POSITION pos =
15        pDoc->GetElements().GetHeadPosition();
16
17    // ** Cat timp elementul nu este NULL, adaugam elemente
18    while(pos)
19    {
20        // ** Accesam urmatorul element din lista
21        CString strElement =
22            pDoc->GetElement().GetNext(pos);
23
24        // ** Il inseram in reprezentarea de tip lista
25        GetListCtrl().InsertItem(0, strElement); — ❷
26    }
27 }
```

❶ Documentul poate fi accesat oricând prin intermediul funcției `GetDocument()` a reprezentării.

❷ Aici se adaugă elementul în listă, prin intermediul funcției `InsertItem()`.

Așa cum se vede în linia 9, un pointer la document se poate obține prin intermediul funcției `GetDocument()` a reprezentării, funcție care întoarce documentul posesor al reprezentării.

Macrodefiniția `ASSERT_VALID()` din linia 11 verifică dacă obiectul indicat de pointer este valid (în cazul de față, este vorba despre documentul `CListVDoc`).

Utilizarea macrodefiniției `ASSERT` pentru interceptarea problemelor înainte de manifestarea acestora

Macrodefinițiile `ASSERT()` și `ASSERT_VALID()` vă ajută la detectarea erorilor într-o fază timpurie a apariției lor. `ASSERT` este utilizat pentru a asigura că o condiție are valoarea `True`. De exemplu, `ASSERT(a>10)` va genera un mesaj de avertisment în cazul în care `a` este mai mic decât 10.

Liniile 14 și 15 folosesc noua funcție de acces, `GetElements()`, pentru a obține lista de șiruri, apelând funcția `GetHeadPosition()` a obiectului `CStringList` cu scopul de a inițializa variabila `pos`, de tip `POSITION`, cu începutul listei. Bucla `while` din linia 17 execută repetat liniile de la 20 la 23 cât timp variabila `pos` nu este zero, condiție care arată că mai există elemente în listă.

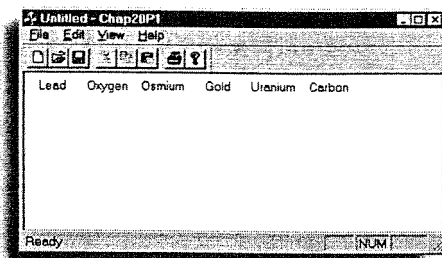
În linia 21 din interiorul buclei, în variabila `strElement` este înscris șirul întors de funcția `GetNext()`, acesta reprezentând elementul următor (sau primul element, dacă ne aflăm la începutul listei). În cazul în care funcția întoarce ultimul element, variabila `pos` va căpăta valoarea zero și bucla `while` nu va mai itera.

În fine, șirul obținut este inserat în cadrul controlului listă prin intermediul funcției `InsertItem()`, aceasta primind ca parametri o poziție (zero, în cazul de față) și șirul de adăugat. Dacă ați remarcat, controlul listă al reprezentării este obținut cu ajutorul funcției `GetListCtrl()`, aceasta fiind analogă cu funcția noastră `GetElements()` prin aceea că întoarce o referință la un *membru privat* cu scopul de proteja modificarea accidentală a membrului respectiv.

Dacă asamblați și rulați aplicația după efectuarea modificărilor din listingul 19.2, veți vedea că reprezentarea de tip listă afișează numele elementelor sub o formă destul de brută, așa cum o înfățișează figura 19.1.

FIGURA 19.1

O reprezentare simplă de tip listă care afișează câteva elemente.



Modificarea stilului unei liste

Aspectul listei de mai sus nu este prea rafinat. Stilul implicit de afișare pentru o reprezentare de tip listă dispune elementele de-a lungul paginii. Dacă doriți să îmbunătățiți aspectul puteți să alegeți un alt stil de listă. Pentru afișarea elementelor pe verticală veți apela la stilul `LVS_LIST`. Există patru stiluri posibile, prezentate în tabelul 19.1.

Selectarea unui întreg rând al controlului listă

În modul raport, selectarea unui element din cadrul controlului duce implicit la evidențierea textului din prima coloană. Prin stabilirea indicatorului `LVS_EX_FULLROWSELECT` puteți să determinați evidențierea întregului rând. Aceasta înseamnă, totodată, că un rând poate fi selectat efectuând clic pe oricare dintre componentele sale.

TABELUL 19.1 Stiluri pentru reprezentarea de tip listă

Stil pentru reprezentarea de tip listă	Descrierea stilului
<code>LVS_LIST</code>	O listă obișnuită, afișată de sus în jos
<code>LVS_REPORT</code>	La fel ca <code>LVS_LIST</code> , dar cu coloane cu antet
<code>LVS_ICON</code>	Pictograme mari, dispuse de la stânga la dreapta, rând cu rând
<code>LVS_SMALLICON</code>	Pictograme mici, dispuse de la stânga la dreapta, rând cu rând

Listingul 19.3 prezintă adăugirile necesare pentru a înlocui stilul cu pictograme implicit (`LVS_ICON`) cu un stil de listă (`LVS_LIST`). Aceasta se face determinând valorile de stil asociate ferestrei, modificându-le și apoi transmițându-le înapoi către fereastră.

LISTINGUL 19.3 LST20_3.CPP – Modificarea stilului implicit al reprezentării de tip listă cu ajutorul funcțiilor `GetWindowLong()` și `SetWindowLong()`

```

1  GetListCtrl().InsertItem(0, strElement);
2  }
3
4  // ** Determinam indicatorii de stil curenti
5  DWORD dwStyle =
6      GetWindowLong(GetListCtrl().GetSafeHwnd(), — ❶
7      GWL_STYLE);
8
9  // ** Anulam stilul curent
10  dwStyle &= ~LVS_TYPEMASK;
11
12  // ** Stabilim stilul lista
13  dwStyle |= LVS_LIST; — ❷
14
15  // ** Transmitem indicatorii înapoi la reprezentarea de tip lista
16  SetWindowLong(GetListCtrl().GetSafeHwnd(), — ❸
17      GWL_STYLE, dwStyle);
18
19  // ** Actualizam afisarea reprezentarii de tip lista
20  SetRedraw(TRUE);
21  }
```

❶ În linia 6 se determină stilul curent al controlului listă.

❷ Aici este modificat stilul.

❸ În linia 16, stilul modificat este trimis înapoi la controlul listă.

Linile prezentate în listingul 19.3 trebuie adăugate la sfârșitul funcției `OnInitialUpdate()`, după bucla `while`. Pentru a putea stabili un stil trebuie mai întâi să obțineți stilul curent al ferestrei, apelând în acest sens funcția `SetWindowLong()`. Specificând indicatorul `GWL_STYLE` arătați că dintre proprietățile ferestrei vă interesează opțiunile de stil. `GetWindowLong()` poate să furnizeze o mulțime de indicatori asociați unei ferestre. Nu trebuie să modificați decât indicatorul `LVS_`, așa că restul combinației de indicatori trebuie conservată. Valoarea întoarsă este depusă în linia 5 într-o variabilă de tip `DWORD`, numită `dwStyle`. Linia 10 anulează apoi indicatorii de stil existenți prin intermediul operatorilor la nivel de bit `&=` și `~` din C++. Prin aplicarea operatorului `~` din C++ asupra valorii `LVS_TYPEMASK`, biții acestei măști sunt inversați (biții 1 devin 0, iar biții 0 devin 1).

Pentru că în această mască sunt stocate toate opțiunile posibile pentru stilul unei reprezentări de tip listă, inversarea sa va avea ca rezultat o mască în care sunt activați toți biții, mai puțin cei corespunzători acestor opțiuni. Apoi, prin intermediul operatorului `&=`, puteți să efectuați un ȘI logic între biții existenți și masca de stil. Efectul este înlăturarea tuturor opțiunii de stil curente pentru reprezentarea de tip listă.

În continuare trebuie să adăugați stilul dorit printr-un SAU logic, folosind operatorul `|=` ca în linia 13. În consecință, indicatorul specificat este adăugat la cei curenți. Odată stabilită combinația de indicatori, trebuie să o trimiteți înapoi ferestrei prin apelul funcției `SetWindowLong()`. `SetWindowLong()` primește ca parametri un indicator către fereastră, un indicator de context (`GWL_STYLE`) și combinația de opțiuni. Acest apel se găsește în linia 16. În fine, apelul `SetRedraw()` din linia 20 determină actualizarea reprezentării de tip listă.

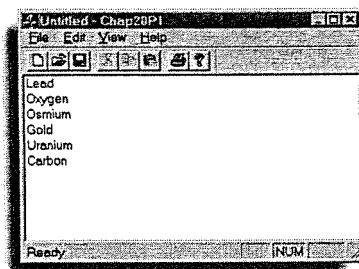
Modificarea culorilor din reprezentarea de tip listă

Puteți să utilizați funcțiile `SetTextColor()` și `SetTextBkColor()` pentru a modifica culoarea textului și a fundalului de text dintr-o reprezentare de tip listă, iar funcția `SetBkColor()` permite modificarea culorii de fundal pentru întreaga fereastră.

După operarea acestor modificări puteți să asamblați și să rulați programul, iar rezultatul va fi o listă de elemente care arată ca în figura 19.2.

FIGURA 19.2

Reprezentarea listei de elemente cu stilul `LVS_LIST`.



VEDEȚI...

- Pentru adăugarea de pictograme și imagini la elementele unei reprezentări de tip listă, vedeți pagina 238.

Adăugarea de coloane și antete asociate

Prin intermediul stilului `LVS_REPORT` puteți adăuga cu ușurință coloane dimensionabile cu antet. Ați putea să faceți mai interesant programul creat, adăugând coloane suplimentare care să afișeze simbolul fiecărui element și numărul său atomic.

Prima modificare necesară este adăugarea în cadrul documentului a noilor informații. Într-o aplicație serioasă ați putea considera ideea creării și utilizării unei clase proprii în scopul colectării acestor informații. Pentru simplitate, însă, noile informații au fost atașate șirurilor existente, fiind separate prin virgulă, așa cum o ilustrează listingul 19.4. Editarea

funcției constructor a documentului se poate face prin efectuarea unui dublu clic pe funcția membru `CListVDoc` care figurează în interiorul clasei `CListVDoc` din cadrul paginii `ClassView` a secțiunii spațiului de lucru al proiectului.

LISTINGUL 19.4 LST20_4.CPP – Adăugarea la șiruri a simbolului și a numărului atomic

```
1 CListVDoc::CListVDoc()
2 {
3     // TODO: add one-time constructor code here
4
5     // ** Numele elementelor si simbolurile acestora
6     m_listElements.AddTail("Carbon,C,6");
7     m_listElements.AddTail("Uranium,U,92");
8     m_listElements.AddTail("Gold,Au,79");
9     m_listElements.AddTail("Osmium,Os,76");
10    m_listElements.AddTail("Oxygen,O,8");
11    m_listElements.AddTail("Lead,Pb,82");
12 }
```

Odată datele modificate, trebuie să inserați antetele de coloană pentru noile informații din cadrul elementelor. Această operație poate fi efectuată în funcția `OnInitialUpdate()` a clasei reprezentare (`CListView`) derivate (din `CListCtrl`), care acum a ajuns destul de întinsă, așa cum se vede în listingul 19.5.

LISTINGUL 19.5 LST 20_5.CPP – Funcția *OnInitialUpdate()*

```
1 void CListView::OnInitialUpdate()
2 {
3     CListCtrl::OnInitialUpdate();
4
5     // TODO: You may populate your ListView with items
6     // its list control through a call to GetListCtrl().
7
8     // ** Inseram coloanele si antetele acestora
9     GetListCtrl().InsertColumn(0,"Element Name",
10                                LVCFMT_LEFT,120);
11     GetListCtrl().InsertColumn(1,"Symbol",
12                                LVCFMT_LEFT,70);
13     GetListCtrl().InsertColumn(2,"Atomic Number",
14                                LVCFMT_LEFT,130);
15
16     // Obținem un pointer la document
17     CListVDoc* pDoc = GetDocument();
18     // ne asiguram ca documentul este valid
19     ASSERT_VALID(pDoc);
20 }
```

(continuare)

LISTINGUL 19.5 Continuare

```

21 // Determinam capul listei de siruri
22 POSITION pos =
23     pDoc->GetElements().GetHeadPosition();
24
25 // Cat timp elementul nu este NULL, adaugam elemente
26 while(pos)
27 {
28     // Accesam urmatorul element din lista
29     CString strElement =
30         pDoc->GetElement().GetNext(pos);
31
32     // ** Extragem din sir numele elementului
33     CString strName =
34         strElement.Left(strElement.Find(","));
35
36     // ** Extragem din sir simbolul si numarul atomic
37     CString strSymbol =
38         strElement.Mid(strElement.Find(",")+1); ❶
39
40     // ** Separam numarul atomic
41     CString strAtomicNumber =
42         strSymbol.Mid(strSymbol.Find(",")+1);
43
44     // ** Detasam numarul atomic de la sfarsit
45     strSymbol =
46         strSymbol.Left(strSymbol.Find(","));
47
48     // ** Inseram numele elementului in lista in coloana 0
49     GetListCtrl().InsertItem(0, strName);
50
51     // ** Inscriem simbolul elementului in a doua coloana
52     GetListCtrl().SetItemText(0, 1,
53         strSymbol);
54
55     // ** Inscriem numarul atomic in a treia coloana
56     GetListCtrl().InsertItem(0, 2,
57         strAtomicNumber);
58
59 }
60
61 // Determinam indicatorii de stil curenti
62 DWORD dwStyle =
63     GetWindowLong(GetListCtrl().GetSafeHwnd(),
64         GWL_STYLE);
65

```

```

66 // Anulam stilul curent
67 dwStyle &= ~LVS_TYPEMASK;
68
69 // ** Stabilim stilul raport
70 dwStyle |= LVS_REPORT; ❷
71
72 // Transmitem indicatorii inapoi la reprezentarea de tip lista
73 SetWindowLong(GetListCtrl().GetSafeHwnd(),
74     GWL_STYLE, dwStyle);
75
76 // Actualizam reprezentarea de tip lista
77 SetRedraw(TRUE);
78 }

```

❶ Această secvență de cod împarte fiecare șir după virgule pentru a extrage elementele fiecărei coloane.

❷ Stilul LVS_REPORT are ca efect adăugarea de coloane cu antet care pot fi dimensionate.

În listingul 19.5, coloanele sunt inserate prin intermediul membrului `InsertColumn()` al controlului listă, în liniile 9, 11 și 13. Această funcție primește numărul coloanei care va fi inserată, numele antetului, un indicator de formatare și lățimea dorită pentru coloană, exprimată în pixeli. Fiecare coloană poate fi formatată astfel încât conținutul său să fie aliniat la stânga, la dreapta sau centrat. În acest sens puteți să transmiteți unul dintre indicatorii `LVC_FMT_LEFT`, `LVC_FMT_RIGHT` sau `LVC_FMT_CENTER`, așa cum se vede în listing.

Stergerea coloanelor

Coloanele inserate pot fi șterse prin intermediul funcției `DeleteColumn()`, transmitând numărul coloanei de șters.

Acum aveți pregătite coloanele, așa că este momentul să inserați elementele corespunzătoare. Dar înainte de aceasta va trebui să faceți puțină chirurgie pe șiruri, în liniile de la 29 la 46. Aceste linii găsesc virgulele și apoi împart fiecare șir în trei șiruri distincte, fiecare conținând informația destinată unei coloane: `strName`, `strSymbol` și `strAtomicNumber` sunt înscrise, respectiv, în cele trei coloane. Observați că pentru prima coloană se folosește tot funcția `InsertItem()` (în linia 49), în timp ce datele celorlalte coloane sunt înscrise cu ajutorul funcției `SetItemText()`, care primește ca parametri poziția elementului, numărul coloanei și informația de înscris.

În fine, trebuie să selectați stilul `LVS_REPORT` (linia 70) care permite afișarea coloanelor inserate, înlocuind vechiul stil `LVS_LIST`.

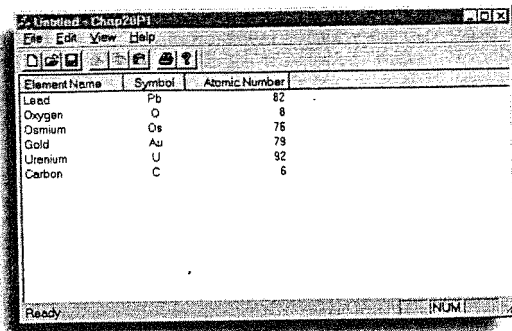
Asamblați și rulați programul după efectuarea acestor modificări și veți vedea lista afișată cu coloanele dimensionabile, așa cum o înfățișează figura 19.3.

VEDEȚI...

➤ Pentru editarea imediată în cadrul reprezentărilor de tip listă, vedeți pagina 447.

FIGURA 19.3

Reprezentarea de tip listă cu coloane.



Determinarea elementelor selectate într-o listă

O utilizare comună a reprezentărilor de tip listă constă în a permite utilizatorului să selecteze unul sau mai multe elemente. Pentru identificarea elementelor după anumite criterii se folosește funcția `GetNextItem()`. Această funcție primește doi parametri, primul reprezentând punctul de început al căutării. Aveți, astfel, posibilitatea de a începe căutarea de la un element arbitrar sau puteți să transmiteți `-1` pentru a porni de la începutul listei. Cel de-al doilea parametru reprezintă o combinație de indicatori prin care se specifică elementele vizate. Acești indicatori sunt prezentați în tabelul 19.2. Precum vedeți, multe relații sunt de natură geometrică, un exemplu fiind `LVNI_TOLEFT`. Motivul este faptul că, în cazul reprezentărilor cu pictograme, a determina care este elementul aflat imediat în stânga unui element anume este o operație foarte delicată. Funcția despre care vorbim acoperă toate aceste ipostaze, inclusiv cazul în care vreți să determinați elementele selectate. Indicatorul `LVNI_SELECTED` determină funcția să găsească următorul element selectat. Atunci când apelați `GetNextItem()` transmițând ca poziție de început ultimul indice întors, funcția va găsi următorul element care satisface criteriile de căutare.

Odată ce aveți un indice furnizat de `GetNextItem()`, conținutul elementului respectiv se poate obține cu ajutorul unei alte funcții a controlului listă, `GetItemText()`. Aceasta primește indicele elementului și coloana vizată, întorcând un `CString` care conține informația solicitată.

TABELUL 19.2 Indicatori folosiți pentru funcția `GetNextItem()`

Indicator	Descriere
<code>LVNI_SELECTED</code>	Elementul este selectat curent
<code>LVNI_FOCUSED</code>	Elementul este încadrat de un dreptunghi punctat care indică focusul
<code>LVNI_ALL</code>	Întoarce elementul următor – acesta este indicatorul implicit
<code>LVNI_ABOVE</code>	Întoarce elementul aflat deasupra celui specificat prin indice
<code>LVNI_BELOW</code>	Întoarce elementul aflat dedesubtul celui specificat prin indice
<code>LVNI_TOLEFT</code>	Întoarce elementul aflat la stânga celui specificat prin indice
<code>LVNI_TORIGHT</code>	Întoarce elementul aflat la dreapta celui specificat prin indice

Ați putea să scrieți cod care să utilizeze aceste funcții pentru a afișa pe bara de titlu a aplicației lista elementelor selectate curent. Mai întâi trebuie să vă gândiți când trebuie actualizată această listă. O alegere evidentă pentru actualizare este momentul în care utilizatorul efectuează un clic pe listă. Este probabil ca selecția să se modifice în acest caz. Puteți să interceptați mesajul generat la efectuarea unui clic pe controlul listă, folosind `ClassWizard` pentru a genera scheletul unei funcții de tratare.

Adăugarea unei funcții de tratare

1. Apăsați `Ctrl+W` pentru a deschide `ClassWizard`.
2. Selectați pagina **Message Maps** și asigurați-vă că în caseta combinată **Class Name** este afișată clasa corespunzătoare (în cazul nostru, `CListView`). În caz contrar, selectați această clasă.
3. Asigurați-vă că clasa `CListView` este selectată în caseta cu listă **Object IDs**.
4. Acum căutați mesajul `=NM_CLICK` în lunga listă de mesaje din caseta **Messages**.
5. Efectuați un dublu clic pe `=NM_CLICK` pentru a apela caseta de dialog **Add Member Function**, aceasta propunând numele `OnClick` pentru noua funcție membru.
6. Efectuați un clic pe **OK** pentru a adăuga noua funcție de tratare.
7. Efectuați un clic pe **Edit Code** pentru a începe să editați noua funcție de tratare.

Acum aveți o funcție de tratare `OnClick()` care va fi apelată de fiecare dată când utilizatorul efectuează un clic undeva în cadrul reprezentării de tip listă.

Pentru determinarea elementelor selectate și afișarea lor pe bara de titlu va trebui să adăugați liniile de cod prezentate în listingul 19.6.

LISTINGUL 19.6 LST20_6.CPP – Determinarea elementelor selectate cu ajutorul funcției `GetNextItem()`

```

1 void CListView::OnClick(NMHDR* pNMHDR,
2   LPARAM* pResult)
3 {
4     // TODO: Add your control notification handler
5
6     *pResult = 0;
7
8     // ** Cream sirul care va contine elementele selectate
9     CString strSelectedItems;
10
11    // ** Primul indice transmis functiei GetNextItem() trebuie sa
12    // fie zero
13    int nSelected=-1;
14    do
15    {
16        // ** Cautam urmatorul element selectat

```

(continuare)

LISTINGUL 19.6 Continuare

```

15     nSelected = GetListCtrl().GetNextItem(
16         nSelected, LVNI_SELECTED);
17
18     // ** Am gasit un element selectat?
19     if (nSelected != -1)
20     {
21         // ** Adaugam continutul sau in sirul creat
22         strSelectedItem += " " +
23             GetListCtrl().GetItemText(nSelected, 0);
24     }
25 } while(nSelected != -1);
26
27 // ** Elementele selectate formeaza titlul documentului
28 GetDocument()->SetTitle(
29     "Selected:" + strSelectedItem);
30 }

```

❶ GetNextItem() caută după criteriile specificate. Aici, LVNI_SELECTED determină căutarea următorului element selectat.

❷ Adăugăm elementul selectat în lista care va conține toate elementele selectate.

Șirul declarat în linia 8, strSelectedItem, este folosit pentru a reține lista elementelor selectate. Observați în linia 11 că nSelected este declarat ca întreg și inițializat cu -1. Această variabilă va fi transmisă ca indice în apelul funcției GetNextItem() din linia 15. În același apel este transmis și indicatorul LVNI_SELECTED, care arată funcției că sunteți interesat în căutarea următorului element selectat.

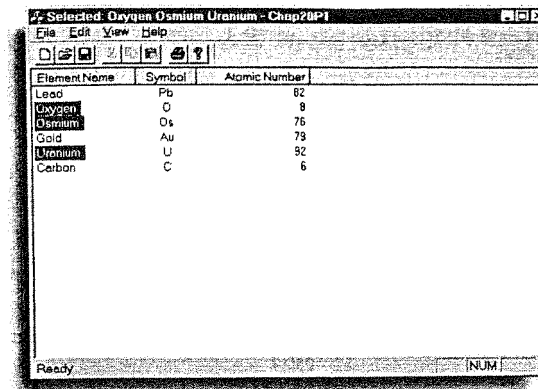
La primul apel, nSelected are valoarea -1, așa că funcția va începe căutarea de la începutul listei. În cazul în care există elemente selectate, indicele următorului dintre acestea va fi întors și înscris în variabila nSelected. Instrucțiunea if din linia 19 verifică dacă nSelected este diferit de -1 (dacă este -1, înseamnă că nu mai există alte elemente selectate). În cazul în care a fost găsit un element selectat, în linia 23 este apelată funcția GetItemText(), fiind transmise un indice și un număr de coloană (0, pentru a specifica prima coloană). Informația furnizată este apoi adăugată (împreună cu un spațiu) în lista construită în șirul strSelectedItem.

Testul while din linia 25 va duce la repetarea întregii bucle do în cazul în care a fost găsit un element selectat. Altfel, toate elementele selectate au fost identificate și puteți să afișați lista lor ca titlu al documentului, așa cum o arată linia 28.

Dacă asamblați și rulați programul după adăugarea codului de mai sus în cadrul funcției de tratare, elementele selectate vor fi afișate pe bara de titlu, după cum se vede în figura 19.4.

FIGURA 19.4

Elementele selectate sunt identificate și afișate pe bara de titlu.



Crearea și utilizarea unei reprezentări de tip arbore

Reprezentările de tip listă sunt excelente pentru afișarea și manipularea listelor simple de elemente. Dar uneori este nevoie de afișarea unor informații ierarhice; în acest caz aveți nevoie de o reprezentare de tip arbore. Un foarte bun exemplu de reprezentare de tip arbore este secțiunea din stânga a aplicației Windows Explorer. Aceasta afișează o mulțime de dosare care pot fi deschise pentru a afișa alte dosare sau fișiere.

VEDEȚI...

- Pentru utilizarea unui control arbore într-o casetă de dialog, vedeți pagina 116.
- Pentru folosirea simultană a mai multor reprezentări, ca în Windows Explorer, vedeți pagina 458.
- Pentru modificarea fontului implicit cu ajutorul funcției SetFont(), vedeți pagina 393.

Crearea unei aplicații cu reprezentare de tip arbore prin intermediul AppWizard

Puteți utiliza AppWizard pentru a genera automat o aplicație SDI care să folosească o reprezentare de tip arbore. Pașii de urmat seamănă foarte mult cu cazul reprezentării de tip listă, cu deosebirea că acolo unde ați selectat clasa CListView veți alege acum clasa CTreeView. Puteți să urmați secvența de pași „Crearea unei infrastructuri SDI cu reprezentare de tip listă”, prezentată mai devreme în acest capitol, înlocuind CListView cu CTreeView. Dacă veți implementa și exemplele de cod, nu uitați să denumiți proiectul Treev.

Modificarea stilului unui arbore

Asemeni reprezentărilor de tip listă, reprezentările de tip arbore folosesc o serie de indicatori de stil. Dar, spre deosebire de cazul listelor, acești indicatori se utilizează pentru adăugarea unor diverse componente într-o reprezentare de tip arbore. Aceste componente sunt liniile care unesc elementele părinte și elementele subordonate și butoanele care deschid și închid nivelele ierarhice. Există, de asemenea, indicatori care dezactivează

tragerea și plasarea sau care permit editarea directă a elementelor. O listă a acestor indicatori este înfățișată în tabelul 19.3.

TABELUL 19.3 Indicatori de stil pentru reprezentarea de tip arbore

Indicator	Descriere
TVS_HASLINES	Trasează linii între elementele subordonate și părinții acestora
TVS_LINESATROOT	Trasează linii între nodul rădăcină și elementele subordonate acestuia
TVS_HASBUTTONS	Afișează mici butoane care permit expandarea sau condensarea nivelelor ierarhice
TVS_SHOWSELALWAYS	Evidențiază elementul selectat chiar dacă este activată o altă fereastră
TVS_DISABLEDROAGDROP	Dezactivează tragerea elementelor din arbore în cadrul operațiilor de tragere și plasare

Majoritatea arborilor combină mai mulți dintre acești indicatori de stil pentru a obține o reprezentare arborescentă în stilul Explorer. Ca și în cazul reprezentărilor de tip listă, stabilirea stilurilor se face apelând funcția `GetWindowLong()` pentru a determina opțiunile curente, modificând biții corespunzători și transmițând noile opțiuni prin apelul `SetWindowLong()`. În listingul 19.7, observați că în liniile de la 55 la 63 se stabilesc opțiunile de stil date de combinația `TVS_HASLINES + TVS_HASBUTTONS + TVS_LINESATROOT`, efectul fiind conectarea cu linii a elementelor subordonate, afișarea butoanelor de expandare și condensare și, respectiv, conectarea cu linii a elementelor rădăcină.

Selectarea unui întreg rând al controlului arbore

În mod implicit, la selectarea unui element dintr-un arbore este evidențiată exclusiv eticheta acestuia. Prin stabilirea indicatorului `TVS_FULLROWSELECT` puteți să determinați evidențierea întregului rând, inclusiv a imaginilor prezente. Rețineți că acest stil și stilul `TVS_HASLINES` se exclud reciproc.

Inserarea de elemente

Cel mai bine este să priviți o reprezentare de tip arbore ca o listă de liste expandabile. Controlul se obține prin intermediul funcției de acces `GetTreeCtrl()` a reprezentării. Odată determinat controlul, adăugarea unui element se efectuează prin apelul funcției `InsertItem()` a acestuia. `InsertItem()` are mai multe forme, fiecare primind diferiți parametri; cea mai simplă formă primește doar trei parametri. Acești parametri sunt șirul care trebuie inserat, un indicator către părintele noului element și un indicator către poziția după care va fi inserat elementul. Obligatoriu este doar șirul de inserat; ceilalți doi parametri au valori implicite care determină adăugarea șirului la nivelul rădăcină, la sfârșitul listei. Evident, nu are rost să lucrați astfel, deoarece ceea ce rezultă nu este un arbore, ci o listă.

Asocierea elementelor dintr-un arbore cu datele aplicației

Uneori este de preferat să asociați fiecare element dintr-un arbore cu un pointer la un obiect. În acest sens veți folosi funcțiile `SetItemData()` și `GetItemData()`.

Funcția `InsertItem()` întoarce un indicator către elementul nou inserat. Acest indicator poate fi utilizat în operații de inserare ulterioare, ca părinte sau ca element după care să se efectueze inserarea. Atunci când începeți să creați arborele, primul element trebuie să aibă ca părinte `TVI_ROOT`; acest indicator precizează că părintele elementului este rădăcina arborelui. Operații ulterioare de inserare pot să specifice ca părinte un element existent, caz în care se formează o ramură și noul element devine subordonat elementului inserat anterior.

Cel de-al treilea parametru al funcției `InsertItem()` vă permite să specificați locul în care se efectuează inserarea; în mod implicit, acest parametru are valoarea `TVI_LAST`, care determină plasarea noului element la sfârșitul listei. Puteți, însă, să specificați un element inserat anterior, iar noul element va fi inserat după acesta, sau puteți să folosiți indicatorul `TVI_SORT`, care plasează noul element astfel încât nivelul curent să fie ordonat alfabetic.

Stergerea de elemente dintr-un arbore

Stergerea elementelor inserate într-un arbore se face cu ajutorul funcției membru `DeleteItem()`; această funcție primește ca parametru un indicator `HTREEITEM` către elementul vizat.

Puteți să folosiți toți acești indicatori pentru a insera elemente în diferite poziții în cadrul unei reprezentări de tip arbore, așa cum o ilustrează codul din listingul 19.7. La generarea proiectului, AppWizard a creat o funcție `OnInitialUpdate()`. Puteți să folosiți această funcție pentru a insera elemente în reprezentarea de tip arbore chiar înainte de afișarea acesteia.

LISTINGUL 19.7 LST20_7.CPP – Inserarea de elemente în cadrul unei reprezentări de tip arbore

```

1 void CTreeView::OnInitialUpdate()
2 {
3     CTreeView::OnInitialUpdate();
4
5     // TODO: You may populate your TreeView with items
6     // its tree control through a call to GetTreeCtrl()
7
8     // ** Cream o scurtatura catre controlul arbore
9     CTreeCtrl& tree = GetTreeCtrl();
10
11    // ** Inseram un element la nivelul radacina
12    HTREEITEM hAnimals = tree.InsertItem("Animals");
13
```

(continuare)

LISTINGUL 19.7 Continuare

```

14 // ** Inseram un element subordonat elementului radacina
15 HTREEITEM hVerts =
16     tree.InsertItem("Vertibrates",hAnimals);
17
18 // ** Inseram elemente subordonate elementului hVerts
19 tree.InsertItem("Whales",hVerts,TVI_SORT);
20 tree.InsertItem("Dogs",hVerts,TVI_SORT);
21 tree.InsertItem("Humans",hVerts,TVI_SORT);
22
23 // ** Inseram un element subordonat elementului radacina
24 HTREEITEM hInverts =
25     tree.InsertItem("Invertibrates",hAnimals);
26
27 // ** Inseram elemente subordonate elementului hInverts
28 tree.InsertItem("Jellyfish",hInverts,TVI_SORT);
29 tree.InsertItem("Worms",hInverts,TVI_SORT);
30 tree.InsertItem("Snails",hInverts,TVI_SORT);
31
32 // ** Inseram un element la nivelul radacina, dupa hAnimals
33 HTREEITEM hPlants =
34     tree.InsertItem("Plants",TVI_ROOT,hAnimals);
35
36 // ** Inseram un element subordonat noului element radacina
37 HTREEITEM hFruit =
38     tree.InsertItem("Fruit", hPlants);
39
40 // ** Inseram elemente subordonate elementului hFruit
41 tree.InsertItem("Apples",hFruit,TVI_SORT);
42 tree.InsertItem("Plums",hFruit,TVI_SORT);
43 tree.InsertItem("Pears",hFruit,TVI_SORT);
44
45 // ** Inseram un element subordonat noului element radacina
46 HTREEITEM hCereal =
47     tree.InsertItem("Cereal", hPlants);
48
49 // ** Inseram elemente subordonate elementului hCereal
50 tree.InsertItem("Wheat", hCereal,TVI_SORT);
51 tree.InsertItem("Rye", hCereal,TVI_SORT);
52 tree.InsertItem("Rice", hCereal,TVI_SORT);
53
54 // ** Determinam indicatorii de stil curenti
55 DWORD dwStyle =
56     GetWindowLong(GetTreeCtrl().GetSafeHwnd(),
57                     GWL_STYLE);
58

```

```

59 // ** Stabilim stilurile dorite — ①
60 dwStyle |= TVS_HASLINES +
        TVS_HASBUTTONS + TVS_LINESATROOT;
61
62 // ** Transmitem indicatorii de stil reprezentarii arbore
63 SetWindowLong(GetTreeCtrl().GetSafeHwnd(),
64               GWL_STYLE,dwStyle);
65
66 // ** Actualizam afisarea reprezentarii de tip arbore
67 SetRedraw(TRUE);
68 }

```

① Stabilim indicatorii de stil.

În linia 9 din listingul 19.7 se creează un obiect `tree` care reprezintă o referință `CTreeCtrl`. În acest fel definim o scurtătură care ne scutește de a apela de fiecare dată `GetTreeCtrl()` pentru a accesa controlul arbore al reprezentării.

Pentru primul element, inserat în linia 12, este suficientă specificarea unui singur parametru, "Animals". Al doilea parametru rămâne la valoarea implicită, `TVI_ROOT`, care este exact ceea ce dorim. Al treilea parametru este implicit `TVI_AFTER`; pentru că ne aflăm la primul element, poziția de inserare nu mai contează.

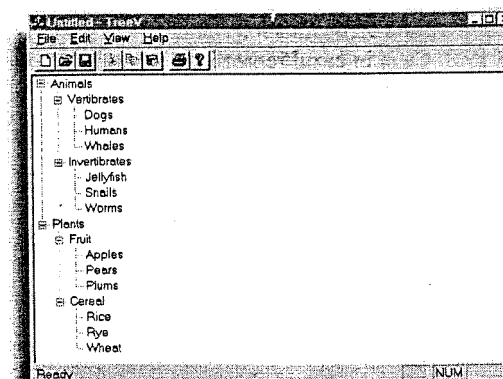
La inserarea elementului "Vertibrates", în linia 16, se specifică `hAnimals` ca element părinte – iar noul element va deveni subordonat elementului `hAnimals`.

În liniile 19-21 sunt inserate animalele vertebrate. Observați că elementul părinte este specificat acum prin indicatorul `hVerts`, iar indicatorul `TVI_SORT` transmis pentru precizarea poziției duce la ordonarea alfabetică a elementelor inserate.

Liniile de la 23 la 52 inserează și alte elemente; diferitele noduri părinte și subordonate vor alcătui o reprezentare ierarhică de tip arbore. Indicatorii de stil necesari sunt stabiliți în liniile de la 56 la 66, așa cum discutăm mai devreme.

FIGURA 19.5

Reprezentarea de tip arbore afișată expandat.



Determinarea nodului selectat

Nodul selectat în cadrul unei reprezentări de tip arbore se poate determina cu ajutorul funcției membru `GetSelectedItem()` a controlului arbore, această funcție întorcând indicatorul `HTREEITEM` al elementului selectat. Ca alternativă, există o funcție `GetNextItem()` care operează la fel ca omonimul său din cadrul reprezentărilor de tip listă. Versiunea oferită de controlul arbore primește doi parametri: primul este un `HTREEITEM`, iar al doilea este o combinație de indicatori prin care specificați criteriile de căutare. Valorile de indicatori posibile sunt prezentate în tabelul 19.4. `GetNextItem()` va întoarce un indicator către elementul găsit sau `NULL` dacă nu găsește nici un element.

TABELUL 19.4 Indicatorii pentru funcția `GetNextItem()` a controlului arbore

Indicator	Descriere
TVGN_CARET	Întoarce elementul selectat curent
TVGN_ROOT	Întoarce elementul rădăcină pentru elementul specificat
TVGN_PARENT	Întoarce părintele elementului specificat
TVGN_CHILD	Întoarce primul element subordonat; elementul specificat trebuie să fie <code>NULL</code>
TVGN_NEXT	Întoarce următorul element subordonat
TVGN_PREVIOUS	Întoarce precedentul element subordonat
TVGN_PREVIOUS_VISIBLE	Întoarce primul element vizibil care precede elementul specificat
TVGN_FIRST_VISIBLE	Întoarce primul element vizibil

Selectarea unui element se poate face și prin program, apelând funcția `SelectItem()` și transmițându-i indicatorul `HTREEITEM` către elementul pe care vreți să-l selectați. Pornind de la un indicator `HTREEITEM` întors de un apel `GetSelectedItem()` sau `GetNextItem()`, eticheta asociată elementului respectiv poate fi obținută cu ajutorul funcției `GetItemText()`.

Codul care determină elementul selectat curent și eticheta asociată acestuia este prezentat în listingul 19.8. Ca și în cazul reprezentării de tip listă, *mesajul de notificare reflectat* `TVN_SELCHANGED` este interceptat prin funcția de tratare `OnSelChanged()` a clasei reprezentare, funcție care poate fi creată în cadrul paginii **Message Maps** din caseta de dialog **ClassWizard**.

LISTINGUL 19.8 LST20_8.CPP – Determinarea etichetei asociate elementului selectat curent

```
1 void CTreeView::OnSelChanged(NMHDR* pNMHDR,
  ↳LRESULT* pResult)
2 {
3     NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
4
```

```
5 // ** Obținem un indicator pentru elementul selectat
6 HTREEITEM hSelected =
7     GetTreeCtrl().GetSelectedItem();
8
9 // ** Ne asigurăm ca exista un element selectat
10 if (hSelected >= 0)
11 {
12     // ** Obținem eticheta asociată elementului selectat
13     CString strSelected =
14         GetTreeCtrl().GetItemText(hSelected);
15
16     // ** Afisăm elementul selectat pe bara de titlu a ferestrei
17     GetDocument()->SetTitle(strSelected);
18 }
19
20 *pResult = 0;
21 }
```

Funcția `GetSelectedItem()` este folosită pentru a obține un indicator către elementul selectat, în linia 6 a listingului 19.8, validitatea acestui indicator fiind apoi verificată în linia 9. Dacă totul este în regulă, apelul `GetItemText()` din linia 14 întoarce eticheta asociată elementului selectat, aceasta devenind titlul documentului în linia 17.

Dacă adăugați acest cod și apoi asamblați și rulați programul, veți vedea că elementul selectat este afișat pe bara de titlu.

Controlul editării imediate

Probabil că ați întâlnit aplicații care vă permit să modificați conținutul unei reprezentări de tip arbore efectuând un clic pe eticheta unui element și modificând-o pe loc. Această facilități se numește *editare imediată*. Implementarea sa este destul de facilă, fiind nevoie de stabilirea indicatorului de stil `TVS_EDITLABELS` și de tratarea a două mesaje de notificare. Mesajele în cauză sunt `TVN_BEGINLABELEDIT` și `TVN_ENDLABELEDIT`, pentru care puteți să adăugați cu **ClassWizard** funcțiile de tratare corespunzătoare, `OnBeginlabeledit()` și `OnEndlabeledit()`.

Editarea imediată în reprezentările de tip listă

Acelasi tip de editare imediată poate fi implementat și într-o reprezentare de tip listă, stabilind în acest sens indicatorul de stil `LVS_EDITLABELS`.

Odată stabilit stilul `TVS_EDITLABELS` pentru o reprezentare de tip arbore, utilizatorul va putea să editeze orice etichetă din arbore prin efectuarea unei secvențe de două clicuri pe suprafața acesteia. În consecință va fi apelată funcția de tratare `OnBeginlabeledit()`, iar aici puteți să utilizați funcția `GetEditControl()` pentru a manipula controlul de editare care a fost adăugat dinamic la arbore cu scopul de a permite editarea pe loc. Pointerul la

controlul de editare poate fi folosit pentru a stabili elemente precum culoarea textului sau pentru a limita dimensiunea textului introdus.

Utilizatorul poate apoi să modifice eticheta, apăsând Enter sau efectuând un clic în afara controlului de editare pentru a încheia editarea. Este momentul în care va fi apelată funcția de tratare `OnEndlabeledit()`. Acum trebuie să apelați din nou la funcția `GetEditControl()` pentru a accesa încă o dată controlul de editare, de data aceasta cu scopul de a determina eticheta modificată. În acest moment puteți să validați noua etichetă și, eventual, să o înregistrați în arbore. Dacă nu înregistrați acum în cadrul arborelui eticheta modificată, operația de editare nu va avea nici un efect, iar elementul în cauză își va păstra vechea etichetă. Pentru înregistrarea în arbore a noii etichete puteți apela la funcția `SetItemText()`.

Adăugarea stilului `TVS_EDITLABELS` alături de celelalte stiluri ale controlului arbore se poate face astfel:

```
// ** Stabilim indicatorii de stil pentru arbore
dwStyle |= TVS_HASLINES + TVS_HASBUTTONS + TVS_LINESATROOT
           + TVS_EDITLABELS; // ** Permite editarea etichetelor
```

Implementările funcțiilor de tratare `OnBeginlabeledit()` și `OnEndlabeledit()` sunt prezentate în listingul 19.9.

Începerea editării imediate

Începerea editării imediate într-o reprezentare de tip arbore este puțin mai delicată; nu trebuie să efectuați un dublu clic pe elementul vizat, ci o secvență de două clicuri cu o pauză între ele.

Dacă adăugați aceste funcții de tratare și modificați corespunzător stilul controlului arbore, după asamblarea și rularea aplicației veți putea să editați etichetele arborelui prin efectuarea unei secvențe de două clicuri, așa cum se vede în figura 19.6.

LISTINGUL 19.9 LST20_9.CPP – Implementarea funcțiilor de tratare `OnBeginlabeledit()` și `OnEndlabeledit()` pentru editarea imediată

```
1 void CTreeView::OnBeginlabeledit(NMHDR* pNMHDR,
2                                 LRESULT* pResult)
3 {
4     TV_DISPINFO* pTVDispInfo = (TV_DISPINFO*)pNMHDR;
5     // TODO: Add your control notification handler
6
7     GetTreeCtrl().GetEditControl()->LimitText(20); ①
8
9     *pResult = 0;
10 }
11
12 void CTreeView::OnEndlabeledit(NMHDR* pNMHDR,
13                               LRESULT* pResult)
```

```
14 TV_DISPINFO* pTVDispInfo = (TV_DISPINFO*)pNMHDR;
15 // TODO: Add your control notification handler
16
17 // ** Obținem eticheta modificata de la controlul de editare
18 CString strText;
19 GetTreeCtrl().GetEditControl()->
20     GetWindowText(strText);
21
22 // ** Aici putem efectua o validare a etichetei
23
24 // ** Ne asiguram ca sirul nu este vid
25 if (strText.GetLength()>0)
26 {
27     // ** Determinam indicatorul asociat elementului selectat
28     HTREEITEM hSelected = pTVDispInfo->item.hItem;
29
30     // ** Inscrinem eticheta modificata ②
31     GetTreeCtrl().SetItemText(hSelected, strText);
32 }
33
34 *pResult = 0;
35 }
```

① Limitarea dimensiunii textului introdus în controlul de editare.

② Eticheta modificată este înscrisă în cadrul arborelui.

Funcția de tratare `OnBeginlabeledit()` este destul de simplă și singurul cod adăugat este cel din linia 7, în care textul introdus în controlul de editare este limitat la maximum 20 de caractere.

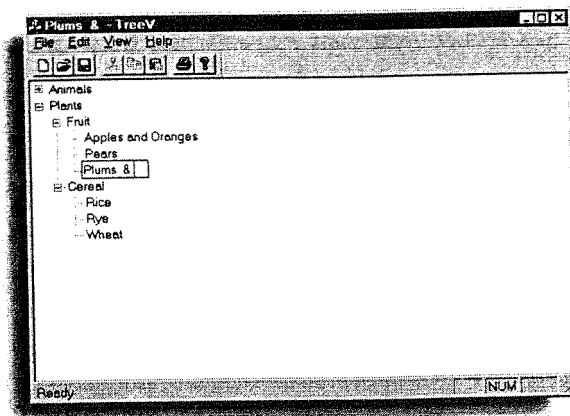
`OnEndlabeledit()` este ceva mai complexă; liniile 19 și 20 determină textul aflat în controlul de editare. Dacă lungimea acestuia nu este nulă (testul din linia 25), obținem elementul selectat curent pe baza pointerului la structura `TV_DISPINFO`, pointer care este transmis funcției de tratare. Indicatorul către elementul selectat este folosit apoi în linia 31 pentru a stabili ca nouă etichetă textul introdus în controlul de editare.

Crearea și utilizarea unei reprezentări de tip editare formatată

O reprezentare de tip editare formatată vă permite să oferiți utilizatorului o interfață de editare a textului foarte puternică și adaptabilă cu un efort de programare foarte mic. Reprezentarea este *activă OLE*, ceea ce înseamnă că puteți să inserați în text imagini, clipuri video și secvențe de sunet. Acesta este un subiect vast și nu îl putem acoperi aici, dar sper să vă puteți forma o imagine.

FIGURA 19.6

Editarea imediată în cadrul unei reprezentări de tip arbore.



Crearea unei reprezentări de tip editare formatată

Puteți să folosiți AppWizard pentru a crea o aplicație SDI a cărei clasă reprezentare să fie derivată din `CRichEditView`, așa cum ați procedat și pentru reprezentările de tip listă și arbore. În mod analog, în ultima casetă de dialog AppWizard trebuie să selectați `CRichEditView` din caseta combinată cu clase de tip reprezentare.

Creați o aplicație SDI numită `RichV` pentru a experimenta facilitățile oferite de o reprezentare de tip editare formatată.

După ce efectuați clic pe **Finish**, va fi afișată o casetă de dialog care vă propune să efectuați un clic pe **OK** pentru a prevedea proiectul cu suport de container OLE. Efectuați clic pe **OK** pentru a adăuga acest suport, deoarece reprezentarea de tip editare formatată operează ca și container OLE. Puteți evita această casetă de dialog selectând opțiunea **Container** la rubrica **What Compound Document Support Would You Like to Include?**, din caseta MFC AppWizard – Step 3.

Reprezentarea de tip editare

Există o reprezentare de tip editare care este mai simplă și are la bază un control de editare multi-linie. Nu dispune de toate facilitățile de editare și de suportul OLE pe care le oferă reprezentarea de tip editare formatată, dar poate fi mai potrivită atunci când nu aveți nevoie de toate aceste elemente. Această reprezentare are la bază clasa `CEditView` și poate fi selectată în ultima casetă de dialog AppWizard.

Acum puteți să asamblați și să rulați aplicația creată fără a mai efectua nici o modificare și veți avea la dispoziție facilități deosebite de editare a textului.

VEDEȚI...

- Despre containerele OLE vom discuta mai amănunțit începând cu pagina 580.

Încărcarea și salvarea textului din cadrul reprezentării

Încercați să introduceți ceva în cadrul reprezentării editare formatată a noii aplicații și apoi efectuați un clic pe meniul **File** și selectați **Save**. Specificați un nume de fișier și salvați textul introdus. Apoi închideți aplicația și lansați-o din nou. Efectuați un clic pe meniul **File** și alegeți opțiunea **Open**, selectați fișierul salvat și efectuați clic pe **OK**. Veți vedea că editorul poate chiar să salveze și să încarce fișiere fără nici o modificare din partea dumneavoastră. Tipărirea și previzualizarea sunt de asemenea implementate. Toate acestea funcționează deoarece containerul OLE îmbină funcționalitatea sa cu cea a aplicației. Atunci când reprezentarea editare formatată este activă, opțiunile din meniul **File** sunt furnizate de reprezentare și sunt implementate automat.

VEDEȚI...

- Pentru mai multe amănunte privind salvarea și încărcarea fișierelor, vedeți pagina 546.
- Pentru o prezentare a containerelor OLE, vedeți pagina 602.

Formatarea paragrafelor

Observați că nu există suport pentru formatarea paragrafelor. Reprezentarea editare formatată oferă în mod standard o serie de funcții de bază pentru procesarea textului, dar lasă în sarcina dumneavoastră implementarea interfeței cu utilizatorul și apelarea acestor funcții.

Puteți să dezvoltați interfața cu utilizatorul pentru a permite formatarea paragrafului curent. Trei noi butoane pe bara cu instrumente vor permite formatarea paragrafelor prin aliniere la stânga, centrare și aliniere la dreapta. Adăugați aceste butoane pe bara cu instrumente standard urmând pașii din secțiunea „Personalizarea barei cu instrumente standard” a capitolului 14, „Utilizarea barelor cu instrumente și a barelor de stare”.

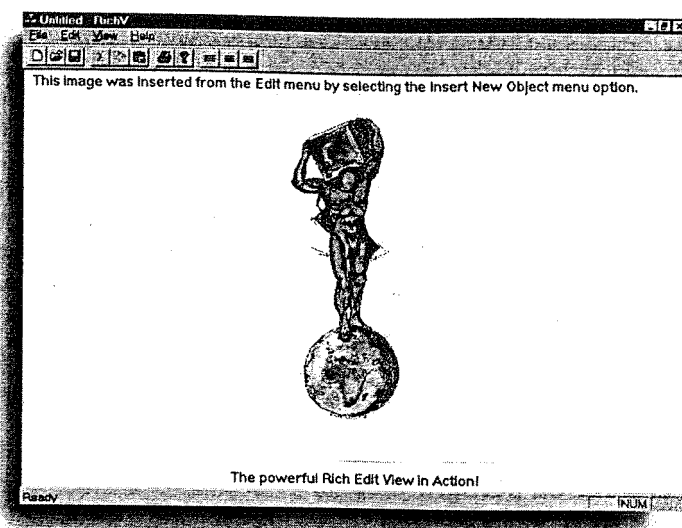
Puteți să concepeți reprezentări grafice proprii pentru aceste opțiuni de aliniere a paragrafelor. Încercările mele prea puțin inspirate în acest sens sunt înfățișate pe bara cu instrumente din figura 19.7. Un prieten mai talentat a creat o superbă imagine bitmap (.bmp) care ilustrează cât de ușor pot fi inserate elemente complexe într-o reprezentare de tip editare formatată. Astfel de imagini pot fi inserate în text efectuând un clic pe meniul **Edit** și selectând opțiunea **Insert New Object**.

Odată adăugate noile butoane pe bara cu instrumente, atribuiți-le identificatorii `ID_ALIGNLEFT`, `ID_ALIGNCENTER` și `ID_ALIGNRIGHT`. Apoi puteți să apelați la ClassWizard pentru a adăuga funcții de tratare corespunzătoare, numite `OnAlignLeft()`, `OnAlignCenter()` și `OnAlignRight()`. Acestea trebuie create în clasa `CRichView`.

Codul pentru formatarea paragrafelor este destul de simplu (așa cum se vede în listingul 19.10). Reprezentarea de tip editare formatată folosește o structură `PARAFORMAT` prin care se specifică mai multe operații de formatare. Membrul `dwMask` este un indicator de tip mască de biți, inițializat corespunzător, ca în linia 6. Operațiile posibile sunt prezentate în tabelul 19.5; acestea privesc în principal indentarea. Diferiții indicatori precizează variabilele membru valide ale structurii `PARAFORMAT`. În acest fel aveți posibilitatea să efectuați operațiile una câte una (ca în exemplu) sau mai multe odată, prin combinarea indicatorilor respectivi.

FIGURA 19.7

O reprezentare de tip editare formatată aflată la lucru.

TABELUL 19.5 Indicatori pentru membrul *dwMask* al structurii *PARAFORMAT*

Indicator	Operația efectuată
PFM_ALIGNMENT	Paragraful selectat este aliniat la stânga, centrat sau la dreapta, în funcție de indicatorul <i>wAlignment</i>
PFM_NUMBERING	Paragraful selectat este numerotat automat, începând cu valoarea membrului <i>wNumbering</i>
PFM_TABSTOPS	Pozițiile de tabulare sunt stabilite pe baza vectorului <i>rgxTabs</i> , numărul lor fiind dat de <i>cTabStops</i>
PFM_OFFSET	Valoarea <i>dxOffset</i> specifică indentarea liniilor relativ la prima linie
PFM_OFFSETINDENT	Valoarea <i>dxStartIndent</i> este o valoare relativă care se adună la indentarea primei linii a paragrafului
PFM_RIGHTINDENT	Valoarea <i>dxRightIndent</i> specifică indentarea față de marginea din dreapta
PFM_STARTINDENT	Valoarea <i>dxStartIndent</i> specifică indentarea primei linii a paragrafului

LISTINGUL 19.10 LST20_10.CPP – Formatarea paragrafelor în cadrul unei reprezentări de tip editare formatată

```

1 void CRichVView::OnAligncenter()
2 {
3     // TODO: Add your command handler code here
4
5     PARAFORMAT pf;
```

```

6     pf.dwMask = PFM_ALIGNMENT;
7     pf.wAlignment = PFA_CENTER; — ❶
8     SetParaFormat(pf);
9 }
10
11 void CRichVView::OnAlignleft()
12 {
13     // TODO: Add your command handler code here
14
15     PARAFORMAT pf;
16     pf.dwMask = PFM_ALIGNMENT;
17     pf.wAlignment = PFA_LEFT; — ❷
18     SetParaFormat(pf);
19 }
20
21 void CRichVView::OnAlignright()
22 {
23     // TODO: Add your command handler code here
24
25     PARAFORMAT pf;
26     pf.dwMask = PFM_ALIGNMENT;
27     pf.wAlignment = PFA_RIGHT; — ❸
28     SetParaFormat(pf);
29 }
```

- ❶ Specificarea indicatorului pentru alinierea centrată a paragrafului.
- ❷ Specificarea indicatorului pentru alinierea la stânga a paragrafului.
- ❸ Specificarea indicatorului pentru alinierea la dreapta a paragrafului.

În liniile 6 și 7, singurul indicator stabilit este *PFM_ALIGNMENT*, fixându-se valoarea variabilei membru corespunzătoare din structură, *wAlignment*. Structura este transmisă apoi funcției membru de formatare, *SetParaFormat()*, aceasta formatând paragraful selectat în funcție de valorile înscrise în structură și de indicatorii care compun membrul *dwMask*, ca în linia 8.

Observați că implementările *OnAlignleft()* și *OnAlignright()* sunt aproape identice, singura deosebire fiind specificarea indicatorilor *PFA_LEFT* și *PFA_RIGHT* în locul indicatorului *PFA_CENTER*.

Dacă asamblați și rulați programul după efectuarea acestor modificări, veți putea să formatați paragrafele selectându-le și efectuând clic pe unul dintre cele trei butoane noi de pe bara cu instrumente.

Inserarea de obiecte OLE

Deoarece reprezentarea de tip editare formatată poate acționa ca un container OLE, puteți să inserați imagini și clipuri prin intermediul opțiunilor de meniu obișnuite. Pentru a încerca,

efectuați un clic pe meniul **Edit** și selectați opțiunea **Insert New Object**. Va fi afișată caseta de dialog **Insert Object**. Observați că există suport pentru o gamă largă de obiecte OLE, oricare dintre acestea putând fi inserat în cadrul reprezentării de tip editare formatată.

VEDEȚI...

➤ Pentru o prezentare a tehnologiilor OLE și COM, vedeți pagina 581.

Crearea și utilizarea unei reprezentări de tip browser HTML

Ca și reprezentarea de tip editare formatată, reprezentarea HTML este foarte puternică și poate dota aplicația cu facilitățile furnizate de browserul de Web Microsoft Internet Explorer 4. Dumneavoastră sunteți responsabil de adăugarea componentelor necesare în interfața cu utilizatorul și de conectarea acestora cu browserul; de aici încolo, totul este deja implementat. Și acest subiect este unul vast, așa că ne vom rezuma la o avangardă.

De unde vine HTML?

HTML este abrevierea pentru HyperText Markup Language. Acesta este limbajul standard folosit la prezentarea informațiilor în cadrul paginilor de Web de pe Internet. Limbajul folosește etichete pentru a specifica aspectele de formatare și permite folosirea graficii și a tabulării.

Crearea unei reprezentări HTML

Puteți să apelați la AppWizard pentru a crea o aplicație a cărei reprezentare să deriveze din **CHtmlView**, selectând această clasă în caseta combinată **Base Class** din ultima casetă de dialog afișată, așa cum ați procedat în cazul reprezentării de tip editare formatată. AppWizard va genera apoi o infrastructură standard în care clasa reprezentare va fi derivată din **CHtmlView**. Secțiunile care urmează vă arată cum puteți utiliza imensa funcționalitate oferită de această clasă reprezentare.

Despre URL

Un URL poate fi adresa de Internet a unei pagini de Web, ca de pildă <http://chaos1.demon.co.uk>. Poate reprezenta, de asemenea, o pagină HTML locală generată într-un program, un fișier text, o arhivă FTP sau orice altă resursă recunoscută de un browser.

Stabilirea adresei URL

Pentru a începe navigarea veți apela funcția **Navigate2()** a clasei **CHtmlView**, transmițând adresa URL dorită.

Browserul va căuta pagina specificată și, dacă este posibil, o va afișa în cadrul reprezentării. Puteți să apelați funcțiile **GoForward()**, **GoBack()**, **GoSearch()**, **GoHome()**, **Stop()** și **Refresh()** pentru a implementa funcționalitatea standard a unui browser ca răspuns la acționarea unor butoane proprii create pe bară cu instrumente. Aceste funcții au același efect cu butoanele corespunzătoare dintr-un browser adevărat.

Există, de asemenea, o funcție **ExecWB()**, care primește diferite comenzi OLE (unele dintre acestea sunt prezentate în tabelul 19.6) în scopul modificării dimensiunilor de font sau al tipăririi paginii curente. Această funcție primește mai mulți indicatori de comandă și o variabilă de tip **ColeVariant** care specifică un parametru necesar.

Spre exemplu, modificarea dimensiunii de font se face prin următorul apel de funcție:

```
ColeVariant vaFontSize(9);
ExecWB(OLECMDID_ZOOM, OLECMDEXEPT_DONTPROMPTUSER,
        & vaFontSize, NULL);
```

ColeVariant este un tip de obiect care poate reține șiruri de caractere, întregi, date calendaristice sau numere zecimale într-o singură variabilă. Astfel de obiecte sunt folosite destul de frecvent în COM și OLE pentru a transmite tipuri de date necunoscute în apelurile de funcții. Funcția **ExecWB()** recunoaște o serie întreagă de comenzi și parametri pe baza cărora efectuează oricare dintre funcțiile unui browser.

TABELUL 19.6 Indicatori de comandă OLE pentru funcția *ExecWB()* a clasei *CHtmlView*

Indicator de comandă OLE	Descriere
OLECMDID_SAVEAS	Salvează pagina curentă
OLECMDID_PRINT	Tipărește pagina curentă
OLECMDID_ZOOM	Stabilește factorul de mărire/micșorare
OLECMDID_FIND	Caută un text în pagina curentă
OLECMDID_STOP	Întrerupe încărcarea paginii
OLECMDID_COPY	Copiază pagina în Clipboard
OLECMDID_ENABLE_INTERACTION	Activează sau dezactivează interacțiunea cu utilizatorul

Tratarea evenimentelor generate de browser

Probabil că veți dori să efectuați anumite operații specifice aplicației la generarea de către browser a diferitelor evenimente, cum ar fi efectuarea de către utilizator a unui clic pe o hiperlegătură. Pentru tratarea acestor evenimente puteți să supradefiniți o serie de funcții.

OnBeforeNavigate2() primește o adresă URL. Puteți să validați această adresă sau să folosiți evenimentul pentru a inițializa un indicator de evoluție (așa cum este vestitul glob care se rotește). Prototipul acestei funcții este

```
void OnBeforeNavigate2(LPCTSTR lpszURL, DWORD nFlags,
    LPCTSTR lpszTargetFrameName, CByteArray& baPostedData,
    LPCTSTR lpszHeaders, BOOL* pbCancel);
```

Spre deosebire de alte funcții virtuale, supradefinirea acestei funcții nu necesită apelul clasei de bază, dar permite întreruperea navigării prin efectuarea în cadrul său a atribuirii `*pbCancel = TRUE`.

Evenimentul de navigare complementar este `OnNavigateComplete2()`, al cărui prototip este

```
virtual void OnNavigateComplete2(LPCTSTR strURL);
```

Dacă în cadrul funcției `OnBeforeNavigate2()` ați inițializat un indicator de evoluție sau o secvență de animație, acesta este locul potrivit pentru a încheia.

Informațiile de stare afișate în mod normal în partea inferioară a ferestrei de browser pot fi manipulate în cadrul funcției `OnStatusTextChanged()`, definită ca

```
virtual void OnStatusTextChange(LPCTSTR lpszText);
```

Observați că parametrul `lpszText` reprezintă un pointer către conținutul noului mesaj de stare.

Funcția `OnProgressChange()` este apelată la fiecare evoluție înregistrată în procesul de încărcare. Funcția virtuală este definită ca

```
virtual void OnProgressChange(long nProgress,  
                              long nProgressMax);
```

Parametrul `nProgress` arată cât de mult s-a încărcat față de totalul `nProgressMax`. Funcția `ExecWB()` vă permite să fixați limita superioară a intervalului de evoluție, transmitând indicatorul `OLECMDID_SETPROGRESSMAX` împreună cu valoarea acestei limite.

Există funcțiile `OnDownloadBegin()` și `OnDownloadEnd()` care sunt apelate la inițierea sau la încheierea operației de descărcare a unui fișier. Puteți folosi aceste funcții pentru a afișa mesaje de avertizare corespunzătoare sau pentru a acapara fișierul descărcat.

Există multe alte funcții pe care le puteți supradefini pentru a răspunde la evenimente, permițându-vă să interacționați cu funcționalitatea browserului pentru a intercepta sau valida comenzi înainte de efectuarea lor și pentru a fi informat de încheierea unei operații. Din păcate, spațiul pe care-l avem la dispoziție aici este prea limitat pentru a acoperi toate acestea, dar sper că v-ați format o idee despre puterea oferită de `CHtmlView`.

CAPITOLUL

20

Crearea de reprezentări multiple

Utilizarea ferestrelor multi-secțiune statice și dinamice

Dezvoltarea de aplicații cu interfață de tip Windows Explorer

Crearea și întreținerea de reprezentări multiple fără utilizarea secțiunilor

Despre reprezentările multiple

Arhitectura Document/Reprezentare aduce numeroase avantaje în procesul de dezvoltare. Cel mai mare dintre avantaje este funcționalitatea dezvoltată pe care o implementează clasele MFC care suportă această arhitectură, efortul depus de programator fiind redus substanțial. Așa cum am arătat în capitole anterioare, aceste clase se ocupă de crearea și administrarea documentelor, a reprezentărilor și a cadrelor. Răspund, de asemenea, de maparea mesajelor de comandă generate de meniuri, de ancorarea barelor cu instrumente, de afișarea barei de stare și de multe alte aspecte.

Separarea codului care gestionează datele (documentul) de codul care implementează interfața cu utilizatorul (reprezentarea) are ca rezultat și o structură flexibilă. Devine posibilă afișarea mai multor reprezentări ale acelorași date. Aceasta poate fi folosită, de exemplu, pentru a afișa simultan două porțiuni distincte dintr-un document sau dintr-un plan de dimensiuni mari, sau pentru a afișa un tabel cu numere alături de un grafic circular al respectivelor numere. Iar două reprezentări nu constituie o limită – unui același document îi pot fi asociate oricâte reprezentări.

Pentru crearea de reprezentări multiple se folosesc mai multe metode. Ferestrele multi-secțiune pot fi utilizate pentru împărțirea unui cadru în secțiuni, fiecare secțiune conținând o reprezentare distinctă. Multe aplicații implementează o astfel de abordare. Windows Explorer folosește o fereastră multi-secțiune pentru a afișa în partea stângă o reprezentare arborescentă a unităților de disc și a dosarelor, în partea dreaptă fiind afișată o reprezentare de tip listă a fișierelor. Mai multe tipuri de controale paginate pot fi utilizate, de asemenea, la implementarea reprezentărilor multiple, permițând utilizatorului să comute între acestea. MS Excel folosește reprezentări paginate pentru a comuta foile de calcul.

Utilizarea ferestrelor multi-secțiune

Utilizarea ferestrelor multi-secțiune este o soluție foarte uzuală pentru afișarea mai multor reprezentări într-o aceeași fereastră cadru. O fereastră multi-secțiune este încapsulată în fereastra cadru și folosită pentru crearea de secțiuni. Fiecare secțiune conține o reprezentare proprie, iar reprezentările pot proveni din aceeași clasă sau din clase diferite. În cazul unei aplicații cu interfață document multiplu, fiecare reprezentare MDI se află în propria fereastră cadru; prin urmare, ferestrele copil MDI pot fi și ele multi-secțiune.

Ferestrele cadru pot fi împărțite pe orizontală, pe verticală sau pe ambele direcții, iar numărul de secțiuni poate fi oricât de mare. Utilizatorul poate să modifice dimensiunile secțiunilor trăgând cu mouse-ul o bară de secționare. Există două tipuri de ferestre multi-secțiune – statice și dinamice. Ambele tipuri sunt implementate de clasa `CSplitterWnd`.

Crearea ferestrelor multi-secțiune dinamice

Un cadru care conține o fereastră multi-secțiune dinamică este împărțit în secțiuni doar atunci când utilizatorul acționează o casetă de secționare aflată în cadrul barei de derulare a ferestrei. Inițial, prima reprezentare (secțiunea stânga-sus) acoperă întreaga zonă client.

După ce caseta de secționare este trasă pe suprafața reprezentării, este afișată o bară de secționare, iar fereastra cadru este împărțită în secțiuni, afișând două sau mai multe reprezentări. Împărțirea în secțiuni duce la construirea obiectelor reprezentare necesare. Secțiunile pot fi înlăturate trăgând o bară de secționare către unul din capetele barei de derulare. La înlăturarea unei secțiuni, obiectul reprezentare corespunzător este distrus.

Secțiunile ferestrelor multi-secțiune dinamice sunt folosite de regulă pentru a afișa două sau mai multe zone diferite ale aceleiași reprezentări și, prin urmare, provin din aceeași clasă reprezentare. Fereastra editorului de text din Developer Studio este un exemplu în acest sens, conținând casete de secționare atât pe orizontală, cât și pe verticală.

Există trei modalități prin care puteți crea ferestre multi-secțiune dinamice în cadrul aplicației. În cazul în care prevedeți de la bun început existența secțiunilor, puteți să le creați cu ajutorul AppWizard. Pentru a adăuga secțiuni dinamice într-o aplicație deja existentă, puteți fie să scrieți codul necesar, fie să inserați componenta **Splitter Bar** (bară de secționare) din galeria de componente și controale. Soluția cea mai simplă rămâne AppWizard. Acesta prezintă avantajul că adaugă automat în meniul **View** o opțiune **Split** care duce la selectarea casetei de secționare, permițând utilizatorului să dimensioneze secțiunile.

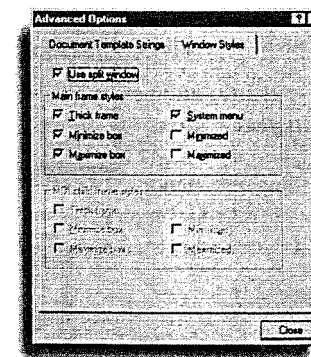
AppWizard creează opțiunea de meniu Split

Dacă selectați opțiunea **Use Split Window** în cadrul AppWizard, în meniul **Window** va fi adăugată o opțiune numită **Split**. În cazul în care adăugați ferestre multi-secțiune într-o aplicație existentă, identificatorul pentru această opțiune trebuie să fie `ID_WINDOW_SPLIT`.

Folosiți AppWizard pentru a crea un nou proiect de tip SDI, numit **DSplit**. În caseta de dialog corespunzătoare pasului 4, efectuați un clic pe butonul **Advanced** și selectați pagina **Window Styles**. Selectați opțiunea **Use Split Window**, așa cum se vede în figura 20.1. În pasul 6, selectați **CScrollView** ca clasă de bază pentru reprezentare.

FIGURA 20.1

Caseta de dialog **Advanced Options** din AppWizard.



Dacă vreți să adăugați ferestre multi-secțiune dinamice într-o aplicație SDI sau MDI existentă, urmați pașii de mai jos. Această secvență este prezentată cu rol informativ; ea nu este necesară pentru exemplul **DSplit**.

Adăugarea componentei bară de secționare

1. Deschideți în Developer Studio proiectul la care vreți să adăugați barele de secționare dinamice.
2. Selectați **Add to Project** din meniul **Project** și selectați opțiunea **Components and Controls** a submeniului afișat. Va fi deschisă caseta de dialog **Components and Controls Gallery**.
3. Efectuați un dublu clic pe catalogul **Visual C++ Components**.
4. Selectați **Splitter Bar** din lista de componente, ca în figura 20.2. În acest moment puteți să efectuați un clic pe butonul **More Info** pentru a vedea informații legate de componenta adăugată.
5. Efectuați un clic pe butonul **Insert**. Efectuați un clic pe butonul **OK** din caseta de mesaj **Insert the Splitter Bar Component**. Este afișată caseta de dialog cu opțiuni **Splitter Bar**.
6. Selectați tipul de secționare dorit; apoi efectuați clic pe **OK**.
7. Închideți caseta de dialog **Components and Controls Gallery**.

VEDEȚI...

- Pentru amănunte privind crearea unei aplicații de tip **SDI**, vedeți pagina 246.
- Pentru informații detaliate despre clasa **CScrollView**, vedeți pagina 415.
- Pentru mai multe informații privind galeria de componente, vedeți pagina 186.

FIGURA 20.2

Caseta de dialog
Components and Controls
Gallery.

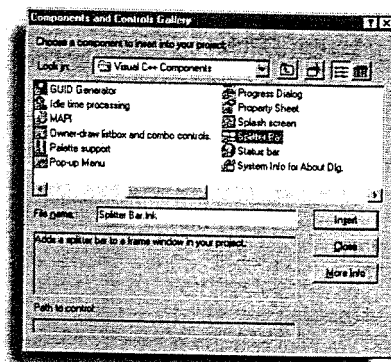
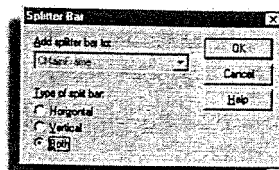


FIGURA 20.3

Caseta de dialog cu opțiuni
privind componenta Splitter
Bar.



Inițializarea ferestrelor multi-secțiune dinamice

Indiferent dacă pentru inserarea ferestrelor multi-secțiune dinamice ați folosit AppWizard sau galeria de componente, codul generat este același. În clasa **CMainFrame** (în cazul aplicației MDI, în clasa **CChildFrame**) este încapsulat un obiect membru de tip **CSplitterWnd**. Obiectul multi-secțiune acaparează zona client a ferestrei cadru și controlează crearea și distrugerea ferestrelor de reprezentare. În exemplul **DSplit** veți găsi următoarea variabilă membru protejată a clasei **CMainFrame**:

```
CSplitterWnd m_wndSplitter;
```

Fereastra multi-secțiune este creată în funcția supradefinită

CMainFrame::OnCreateClient, așa cum o arată listingul 20.1. Această funcție este apelată în timpul procesului de creare a ferestrei cadru, din funcția **CFrameWnd::OnCreate**. Implementarea implicită creează obiectul reprezentare și-i stabilește dimensiunile astfel încât să acopere zona client a ferestrei cadru. Mai întâi este creată fereastra multi-secțiune, iar aceasta se inițializează și creează un obiect reprezentare. Alte reprezentări sunt create în momentul în care utilizatorul secționează fereastra.

LISTINGUL 20.1 LST21_1.CPP – Crearea ferestrei multi-secțiune dinamică în **OnCreateClient**

```
1 BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT /*lpcreatestruct*/,
2                               CCreateContext* pContext) — ❶
3 {
4     return m_wndSplitter.Create(this,
5                                 2, 2,          // TODO: adjust rows and columns
6                                 CSize(10, 10), // TODO: adjust minimum pane size — ❷
7                                 pContext);
8 }
```

❶ Funcția **OnCreateClient** este supradefinită pentru crearea ferestrei multi-secțiune.

❷ La crearea ferestrei multi-secțiune se specifică numărul de linii și de coloane.

Funcția **Create** a ferestrei multi-secțiune (linia 4) poate primi până la șapte parametri. Primul este un pointer la fereastra părinte, **this**. Al doilea și al treilea parametru reprezintă numărul maxim de linii și, respectiv, de coloane (linia 5). Clasa **CSplitterWnd** vă permite să creați cel mult o bară de secționare orizontală și o bară de secționare verticală, ceea ce înseamnă că pentru numărul de linii și de coloane puteți specifica doar 1 sau 2. De exemplu, o linie și două coloane vor duce la crearea doar a unei bare de secționare verticală.

Crearea ferestrelor la momentul execuției

Este ușor să adăugați controale în machetele de dialog, dar pentru a crea la momentul execuției, să spunem, o casetă de editare pe suprafața unei casete de dialog, va trebui să apelați **CEdit::Create**. Prin intermediul funcției **Create** puteți să creați dinamic orice tip de fereastră. Crearea de ferestre la momentul execuției este întotdeauna un proces în două etape. Mai întâi construiți un obiect al clasei fereastră (de exemplu, **CEdit**, **CComboBox**), după care apelați funcția **Create** a acestuia, transmitând stilul ferestrei, dimensiunea inițială, fereastra părinte și un identificator numeric.

Al patrulea parametru este un obiect `CSize` care specifică valorile minime pentru înălțimii și pentru lățimea coloanei (linia 6). Reprezentarea este creată doar dacă dimensiunile secțiunii depășesc aceste limite inferioare, fiind distrusă în momentul în care secțiunea este micșorată sub limite. Aceste valori pot fi modificate la momentul execuției prin intermediul funcțiilor `SetRowInfo` și `SetColumnInfo`. Cel de-al cincilea parametru este pointer la un obiect `CCreateContext`; acest obiect conține informații dinamice despre clasele document, reprezentare și cadru. Structura `pContext` a fost deja inițializată și este transmisă pur și simplu funcției `Create` (linia 7).

Ultimii doi parametri sunt opționali. Cel de-al șaselea permite specificarea opțiunilor de stil pentru fereastră. Combinația implicită este `WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL | SPLS_DYNAMIC_SPLIT`. Ultimul parametru este identificatorul ferestrei copil. Specificați `AFX_IDW_PANE_FIRST`, mai puțin în cazul în care imbricați ferestre multi-secțiune una în cealaltă. Pentru a ilustra utilizarea ferestrelor multi-secțiune dinamice, editați cele două funcții membru ale clasei `CDSplitView` care sunt prezentate în listingul 20.2.

LISTINGUL 20.2 LST21_2.CPP – Exemplu de fereastră multi-secțiune dinamică care sunt afișate 50 de linii de text

```
1 void CDSplitView::OnDraw(CDC* pDC)
2 {
3     CDSplitDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7     TEXTMETRIC tm;
8     int nLineHeight;
9
10    // ** Determinam proprietatile metrice ale fontului curent si
11    // calculam inaltimea unei linii
12    pDC->GetTextMetrics(&tm);
13    nLineHeight = tm.tmHeight + tm.tmExternalLeading; ❶
14
15    // ** Afișam 50 de linii de text
16    CString str;
17    for(int nLine = 1; nLine < 51; nLine++)
18    {
19        str.Format("Line %d - I must NOT feed my homework to my
20        dog.", nLine);
21        pDC->TextOut(5, nLine * nLineHeight, str); ❷
22    }
23
24 void CDSplitView::OnInitialUpdate()
25 {
```

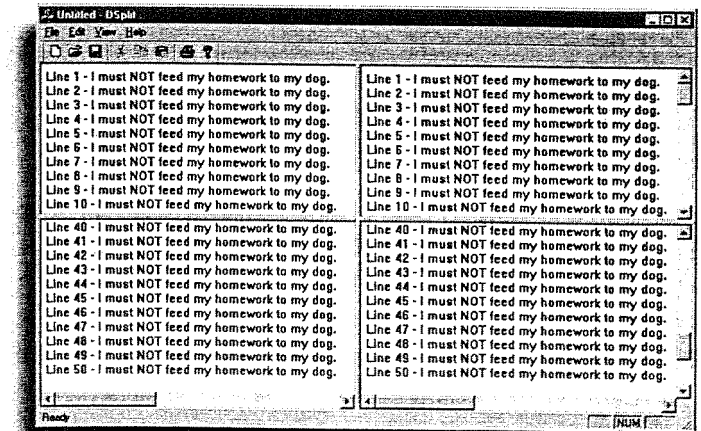
```
25     CScrollView::OnInitialUpdate();
26
27     CSize sizeTotal;
28     // TODO: calculate the total size of this view
29
30     // ** Initializam suprafata totala de derulare la 1000x1000
31     sizeTotal.cx = sizeTotal.cy = 1000;
32     SetScrollSizes(MM_TEXT, sizeTotal); ❸
33 }
```

- ❶ Calculăm înălțimea unei linii de text afișate cu fontul curent.
- ❷ Afișăm textul pe ecran.
- ❸ Stabilirea suprafeței de derulare la 1000x1000 pixeli forțează afișarea barelor de derulare.

Singura modificare adusă funcției `CDSplitView::OnInitialUpdate()` este înlocuirea valorilor 100 din linia 31 cu 1000. Astfel, suprafața logică a reprezentării este mai mare decât suprafața fizică, ceea ce duce la afișarea barelor de derulare. Funcția `OnDraw` supradefinită afișează 50 de linii de text. Pentru a putea calcula înălțimea fontului selectat curent în contextul dispozitiv, linia 7 declară o structură `TEXTMETRIC`, inițializată în linia 11 prin intermediul funcției `CDC::GetTextMetrics`.

Asamblați și rulați aplicația. Selectați casetele de secționare și încercați derularea reprezentărilor. Observați că barele de derulare deserveșc câte două reprezentări. Figura 20.4 înfățișează exemplul `DSplit`.

FIGURA 20.4
Exemplul `DSplit`.



VEDEȚI...

- Pentru mai multe informații despre clasele MDI, vedeți pagina 480.
- Pentru informații privind contextele dispozitiv, vedeți pagina 323.

Crearea de ferestre multi-sectiune statice

Un cadru care conține o fereastră multi-sectiune statică apare împărțit în secțiuni imediat după creare. Numărul de secțiuni, pozițiile lor inițiale și clasele reprezentare sunt specificate la crearea ferestrei cadru. Tot acum sunt create și obiectele reprezentare. Spre deosebire de ferestrele multi-sectiune dinamice, utilizatorul nu poate înlătura secțiunile statice; prin urmare, obiectele reprezentare sunt permanente, fără a mai fi construite și distruse. Bara de secționare este vizibilă întotdeauna și, atunci când este trasă, se blochează în momentul în care o secțiune atinge dimensiunile minime.

Dacă vreți să dezvoltați o aplicație având un aspect similar cu Windows Explorer, care folosește o fereastră multi-sectiune statică ce conține o reprezentare de tip arbore în partea stângă și o reprezentare de tip listă în partea dreaptă, AppWizard vă poate da o mână de ajutor. Așadar, dacă stilul Explorer vă satisface, vedeți secțiunea „Crearea unei aplicații în stilul Windows Explorer”, mai târziu în acest capitol. Singura alternativă pentru crearea de secțiuni statice este scrierea de cod. Pentru a ilustra această modalitate, folosiți AppWizard pentru a crea un nou proiect de tip SDI, numit SSplit. În pasul 6 selectați CEditView ca clasă de bază pentru reprezentare.

Inițializarea ferestrelor multi-sectiune statice

În funcție de aplicație, secțiunile unei ferestre multi-sectiune statice pot afișa diverse tipuri de reprezentări și, în consecință, pot opera cu clase reprezentare diferite. Utilizați AppWizard pentru a crea un nou proiect de tip SDI, numit SSplit. În pasul 6 al AppWizard, selectați CEditView ca clasă de bază pentru reprezentare.

Personalizarea aspectului barelor de secționare

Dacă doriți să personalizați aspectul barelor de secționare, puteți să derivați din CSplitterWnd o clasă proprie și să supradefiniți funcția virtuală OnDrawSplitter(). Funcția primește o valoare de tip enumerare care indică elementul ce trebuie desenat. Aceste elemente sunt splitBox, splitBar, splitIntersection și splitBorder.

În acest proiect veți adăuga codul care implementează o fereastră multi-sectiune statică și o a doua clasă reprezentare. Secțiunea va fi verticală, cu o reprezentare de tip CEditView în secțiunea din stânga și o reprezentare de tip CView în secțiunea din dreapta. După ce ați creat proiectul, urmați pașii de mai jos.

Derivarea unei clase reprezentare proprii cu ajutorul ClassWizard

1. Apăsați Ctrl+W pentru a apela ClassWizard, sau selectați ClassWizard din meniul View.
2. Efectuați un clic pe butonul Add Class și selectați New din lista afișată. Va fi deschisă caseta de dialog New Class.
3. Introduceți numele noii clase (în câmpul Name). Pentru exemplul nostru, introduceți CartView.

4. Selectați CView din caseta combinată Base Class. Apoi efectuați clic pe OK pentru a adăuga noua clasă.
5. Efectuați un clic pe OK pentru a închide caseta de dialog ClassWizard.

Odată creată noua clasă reprezentare, puteți să implementați codul necesar pentru fereastră multi-sectiune statică. Mai întâi, adăugați în clasa CMainFrame o variabilă membru protejată de tip CSplitterWnd. Apoi urmați pașii de mai jos și editați funcția CMainFrame::OnCreateClient, în concordanță cu listingul 20.3.

Implementarea unei ferestre multi-sectiune statică

1. Apăsați Ctrl+W pentru a apela ClassWizard, sau selectați ClassWizard din meniul View.
2. Selectați pagina Message Maps.
3. Selectați CMainFrame în caseta combinată Class Name.
4. Selectați CMainFrame în caseta cu listă Object IDs.
5. Selectați OnCreateClient din caseta cu listă Messages și apoi efectuați un clic pe butonul Add Function.
6. Efectuați un clic pe butonul Edit Code.

LISTINGUL 20.3 LST21_3.CPP – Crearea ferestrei multi-sectiune statică în OnCreateClient

```
1 BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
2     ↳CContext* pContext)
3 {
4     // TODO: Add your specialized code here and/or call the base
5     ↳class
6     // ** Cream fereastra multi-sectiune statica — 1
7     if (!m_wndSplitter.CreateStatic(this, 1, 2))
8         return FALSE;
9
10    // ** Cream doua reprezentari si le atasam la sectiunile create
11    if (!m_wndSplitter.CreateView(0, 0, RUNTIME_
12        ↳CLASS(CSSplitView), CSize(150, 100), pContext) ||
13        m_wndSplitter.CreateView(0, 1, RUNTIME_CLASS(CartView), — 2
14        ↳CSize(100, 100), pContext))
15    {
16        m_wndSplitter.DestroyWindow(); — 3
17        return FALSE;
18    }
19
20    // ** Incheiem cu succes
21    return TRUE;
22 }
```

1 Creăm o fereastră multi-sectiune statică, cu o linie și două coloane (secțiune verticală).

- 2 Creăm câte o reprezentare pentru fiecare secțiune a ferestrei.
- 3 Anulăm totul dacă operația de creare a eșuat.

Rețineți că va trebui să adăugați la începutul fișierului MainFrm.cpp directivele #include următoare:

```
#include "SSplitView.h"
#include "ArtView.h"
```

De asemenea, trebuie să adăugați un antet de declarație pentru clasa document în fișierul SSplitView.h, înainte de definiția de clasă. Această declarație este necesară din cauza ordinii în care sunt incluse acum fișierele antet. Adăugați linia următoare:

```
class CSSplitDoc;
```

deasupra liniei:

```
class CSSplitView : public CEditView
```

Pentru crearea secțiunilor statice este apelată funcția CreateStatic, deoarece Create se utilizează la crearea secțiunilor dinamice.

Funcția CreateStatic a ferestrei multi-secțiune poate primi până la cinci parametri. Primul este un pointer la fereastra părinte, this. Al doilea și al treilea reprezintă numărul maxim de linii și, respectiv, de coloane. Linia 5 specifică o linie și două coloane, rezultatul fiind două secțiuni separate de o bară verticală. Ultimii doi parametri sunt opționali. Cel de-al patrulea vă permite să transmiteți opțiuni de stil pentru fereastră. Ultimul parametru este identificatorul ferestrei copil. Aici ar trebui să specificați valoarea AFX_IDW_PANE_FIRST, excepție făcând cazul în care imbricați ferestre multi-secțiune.

Limitele pentru ferestrele multi-secțiune statice

Spre deosebire de ferestrele multi-secțiune dinamice, care pot avea maximum două rânduri și două coloane, cele statice sunt limitate la 16 rânduri și 16 coloane. Oricum, la atingerea acestei limite interfața nu va mai fi tocmai prietenoasă.

Pentru fiecare secțiune trebuie creat un obiect reprezentare, ceea ce se întâmplă prin apelurile CreateView din liniile 9 și 10. Primii doi parametri sunt linia și coloana în care va fi plasată secțiunea. Al treilea parametru este un pointer la informațiile dinamice ale clasei reprezentare. Această clasă trebuie să fie derivată din CWnd, fiind de obicei derivată direct din CView. Al patrulea parametru este un obiect CSize care specifică valorile minime pentru înălțimea liniei și pentru lățimea coloanei. Aceste valori pot fi modificate la momentul execuției cu ajutorul funcțiilor SetRowInfo și SetColumnInfo. Cel de-al cincilea parametru este un pointer la un obiect CCreateContext. Acesta conține informații dinamice asociate claselor document și cadru. Structura pContext a fost inițializată înainte de a fi transmisă funcției CFrameWnd::OnCreateClient, așa că poate fi transmisă mai departe în apelul CreateView.

În finalul funcției, apelul clasei de bază a fost înlăturat și, în cazul în care crearea secțiunilor și a reprezentărilor asociate a reușit, se întoarce valoarea TRUE (linia 17).

Pentru a ilustra utilizarea secțiunilor statice având asociate clase reprezentare diferite, editați funcția CArtView::OnDraw în concordanță cu listingul 20.4.

LISTINGUL 20.4 LST21_4.CPP – Supradefinirea funcției OnDraw

```
1 void CArtView::OnDraw(CDC* pDC)
2 {
3     CDocument* pDoc = GetDocument(); — 1
4
5     // TODO: add draw code here
6     // ** Salvam pensula curenta
7     CBrush* pOldBrush = pDC->GetCurrentBrush(); — 2
8
9     // ** Cream o pensula plina, de culoare albastra
10    CBrush br;
11    br.CreateSolidBrush( RGB(0,0,255) ); — 3
12
13    // ** Selectam pensula albastra in contextul dispozitiv
14    pDC->SelectObject( &br ); — 4
15    pDC->Ellipse( 1, 1, 300, 300 ); — 5
16    br.Detach(); — 6
17
18    br.CreateHatchBrush( HS_FDIAGONAL, RGB(255,255,0) ); — 7
19    pDC->SelectObject( &br );
20    pDC->Ellipse( 50, 50, 200, 200 ); — 8
21
22    // ** Selectam la loc pensula originala
23    pDC->SelectObject( pOldBrush ); — 9
24 }
```

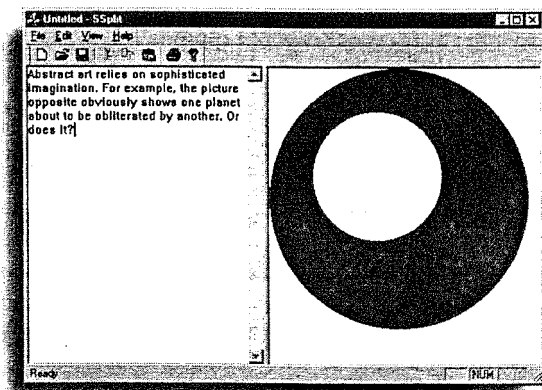
- 1 Apelăm funcția GetDocument() a reprezentării pentru a obține un pointer la obiectul document al aplicației.
- 2 Obținem un pointer la pensula selectată curent în contextul dispozitiv pDC.
- 3 Creăm o pensulă care va desena uniform cu albastru.
- 4 Selectăm noua pensulă în contextul dispozitiv.
- 5 Desenăm elipsa mai mare.
- 6 Detașăm pensula de contextul dispozitiv.
- 7 Creăm o pensulă galbenă care va desena cu linii înclinate la 45 de grade.
- 8 Desenăm elipsa mai mică.
- 9 Selectăm la loc pensula originală din contextul dispozitiv.

Funcția `CartView::OnDraw` este apelată atunci când se impune redesenarea secțiunii din dreapta. Codul desenează două elipse, una cu o pensulă plină și cealaltă cu o pensulă hașurată. Coordonatele trimise ca parametri funcțiilor de desenare, așa cum este `CDC::Ellipse` în liniile 15 și 20, sunt relativi la suprafața reprezentării din fiecare secțiune.

Asamblați și rulați aplicația. Figura 20.5 înfățișează exemplul `SSplit`.

FIGURA 20.5

Exemplul `SSplit`.



Crearea unei aplicații în stilul Windows Explorer

Windows Explorer este un exemplu excelent de aplicație cu reprezentări multiple. O bară verticală separă două secțiuni statice ale căror reprezentări au tipuri diferite – o reprezentare de tip arbore în stânga și o reprezentare de tip listă în dreapta. Această structură generală poate fi utilizată în multe situații. Am folosit acest stil atunci când am dezvoltat un sistem de planificare a producției. Arborele conținea numele angajaților și al echipamentelor, iar în partea dreaptă erau afișate detalii privind activitățile de îndeplinit. AppWizard permite acum generarea directă a unei aplicații în stilul Windows Explorer. Opțiunile necesare în acest scop sunt prezentate în pașii de mai jos.

Crearea unui proiect de tip Explorer

1. Creați un nou proiect MFC AppWizard (exe).
2. În caseta de dialog MFC AppWizard Step 1, selectați unul din butoanele de opțiune **Single Document** sau **Multiple Document**.
3. Asigurați-vă că opțiunea **Document/View Architecture Support?** este validată.
4. În caseta de dialog MFC AppWizard Step 5, selectați butonul de opțiune **Windows Explorer**.

Selectarea acestor opțiuni în cadrul AppWizard va duce la crearea unui schelet de aplicație care conține două clase reprezentare: o clasă `CLeftView` derivată din `CTreeView` și o clasă reprezentare a aplicației derivată din `CListView`. Fereastra multi-secțiune statică este încapsulată automat în fereastra cadru, iar codul necesar pentru crearea reprezentărilor este

generat în funcția supradefinită `OnCreateClient`. Ambele clase reprezentare sunt asociate cu același document și au implementate funcția `GetDocument()`.

În plus, AppWizard adaugă pe bara cu instrumente patru butoane care selectează tipul de afișare în cadrul controlului listă, fiind posibilă afișarea cu pictograme mici, cu pictograme mari, de tip listă sau detaliată. De asemenea, sub bara cu instrumente este adăugată o bară de dialog. Explorer folosește o bară de dialog opțională pentru a afișa un titlu deasupra fiecărei reprezentări, dar dumneavoastră puteți găsi alte întrebări. Dacă doriți să înlăturați bara de dialog, eliminați variabila membru `m_wndDlgBar` a clasei `CMainFrame`, precum și codul din funcția `CMainFrame::OnCreate` care face referire la această variabilă.

În centrul unei reprezentări de tip arbore se află un control arbore

Clasa `CTreeView` este de fapt o clasă de încapsulare pentru controlul arbore, în casetele de dialog fiind folosită versiunea `CTreeCtrl` de încapsulare. Această clasă reprezentare stabilește dimensiunile controlului astfel încât să acopere exact zona client a ferestrei cadru, adaptându-se automat la redimensionarea ferestrei. De asemenea, `CTreeView` permite adăugarea de funcții de tratare pentru comenzile provenite de la meniuri și de la barele cu instrumente. Pentru accesarea controlului arbore încapsulat, apăsați funcția `GetTreeCtrl()`.

Clasa reprezentare `CLeftView` asociată secțiunii din stânga este întotdeauna derivată din `CTreeView` și AppWizard nu permite modificarea acestei opțiuni, însă este posibilă selectarea clasei reprezentare de bază pentru secțiunea din dreapta. În caseta de dialog corespunzătoare pasului 6 din AppWizard, clasa de bază a reprezentării asociate secțiunii din dreapta poate fi selectată dintre clasele aflate în caseta combinată **Base Class**. După generarea scheletului aplicației, va trebui să vă ocupați de popularea controlului arbore și de implementarea clasei reprezentare asociate secțiunii din dreapta.

VEDEȚI...

- Pentru mai multe detalii privind clasa `CTreeView`, vedeți pagina 441.
- Pentru mai multe detalii privind clasa `CListView`, vedeți pagina 427.

Crearea dinamică a reprezentărilor multiple

Arhitectura Document/Reprezentare oferă o flexibilitate nelimitată în ceea ce privește modul de prezentare a datelor către utilizator. Ați văzut deja cum pot fi folosite ferestrele multi-secțiune, dar acestea nu sunt întotdeauna ideale, mai ales dacă există multe reprezentări vizuale diferite ale datelor. Infrastructura MFC se dovedește versatilă, oferind posibilitatea de a construi dinamic o reprezentare și de a o asocia cu un document existent. Odată stabilită asocierea, noua reprezentare interacționează cu documentul la fel ca o reprezentare declarată în macheta de document.

Adăugarea și înlăturarea reprezentărilor

Reprezentările pot fi construite în orice moment al execuției. Stabilirea asocierii dintre un document și un obiect reprezentare nou construit se face prin transmiterea către funcția `CDocument::AddView` a unui pointer la obiectul reprezentare. Anularea asocierii dintre un document și o reprezentare se face apelând funcția `CDocument::RemoveView`, transmițând

și de această dată un pointer la obiectul reprezentare în cauză. Vă atrag atenția că aceste funcții nu instanțiază și nici nu șterg un obiect reprezentare. Acesta trebuie să existe înainte de apelul `AddView` și trebuie distrus numai după apelul `RemoveView`. Documentul întreține o listă a reprezentărilor asociate; cele două funcții pur și simplu manipulează această listă.

O reprezentare este asociată unui singur document

O reprezentare nu poate fi asociată decât cu un singur document; încercarea de a efectua un nou apel `AddView` în care să transmiteți un pointer la aceeași reprezentare va duce la încălcarea unei aserțiuni în cadrul codului MFC.

Atunci când reprezentările sunt create prin intermediul machetelor de document, `AddView` și `RemoveView` sunt apelate din cadrul funcțiilor `CView::OnCreate` și, respectiv, `CView::~CView`. Ambele apelează funcția virtuală `OnChangedViewList` după execuția propriilor implementări. `CDocument::OnChangedViewList` verifică dacă lista de reprezentări a documentului este vidă, caz în care închide documentul printr-un apel `OnCloseDocument`. Dacă preferați un alt comportament puteți să supradefiniți funcția `OnChangedViewList` – de exemplu, ați putea dori ca documentul să rămână deschis chiar dacă nu mai există reprezentări asociate.

Controlul asupra creării și activării reprezentării

Este ușor să controlați când și care reprezentări sunt afișate utilizatorului, prin manipulare funcțiilor aflate la baza arhitecturii Document/Reprezentare. Acestea vă permit să adăugați și să înlăturați reprezentări ca răspuns la acțiunile utilizatorului, oferind posibilitatea de personalizare a interfeței. În secțiunea de față veți crea o aplicație care permite utilizatorului să comute între două reprezentări, cea selectată acoperind întreaga zonă client. Pentru început, folosiți `AppWizard` pentru a crea un nou proiect de tip SDI, numit `VPick`; în pasul 6 selectați `CEditView` ca clasă de bază a reprezentării. Exemplul `VPick` se bazează pe un exemplu anterior, `SSplit`, având scopul de a ilustra o abordare alternativă la utilizarea secțiunilor. Odată generat proiectul, urmați secvența de pași „Derivarea unei clase reprezentare proprii cu ajutorul `ClassWizard`”, secvență pe care ați întâlnit-o mai devreme în acest capitol.

Utilizatorul va selecta reprezentarea dorită dintr-un meniu. Adăugați în meniul `View` două opțiuni corespunzătoare și fixați etichete sugestive, stabilind identificatorii `ID_SHOW_ARTVIEW` și `ID_SHOW_EDITVIEW`. Acum adăugați funcții de tratare pentru comenzile generate de cele două opțiuni. În acest exemplu de tip SDI, funcțiile de tratare comenzilor vor fi implementate în `CMainFrame`; în cazul unui proiect MDI, clasa de implementare ar fi `CChildFrame`.

Adăugarea funcțiilor de tratare a comenzilor de meniu

1. Apăsăți `Ctrl+W` pentru a apela `ClassWizard` sau selectați `ClassWizard` din meniul `View`, iar apoi accesați pagina `Message Maps`.
2. Selectați `CMainFrame` în caseta combinată `Class Name`.
3. Selectați `ID_SHOW_EDIT` în caseta cu listă `Object IDs`.

4. Selectați `COMMAND` din caseta cu listă `Messages`, după care efectuați un clic pe butonul `Add Function`. Efectuați un clic pe butonul `OK` din caseta de dialog `Add Member Function`.
5. Selectați `UPDATE_COMMAND_UI` din caseta cu listă `Messages`, după care efectuați un clic pe butonul `Add Function`. Efectuați un clic pe butonul `OK` din caseta de dialog `Add Member Function`.
6. Selectați `ID_SHOW_ART` în caseta cu listă `Object IDs`.
7. Selectați `COMMAND` din caseta cu listă `Messages`, după care efectuați un clic pe butonul `Add Function`. Efectuați un clic pe butonul `OK` din caseta de dialog `Add Member Function`.
8. Selectați `UPDATE_COMMAND_UI` din caseta cu listă `Messages`, după care efectuați un clic pe butonul `Add Function`. Efectuați un clic pe butonul `OK` din caseta de dialog `Add Member Function`.
9. Efectuați un clic pe `OK` pentru a închide caseta de dialog `ClassWizard`.

Funcțiile `OnShowEdit` și `OnShowArt` au în comun cea mai mare parte a codului. Decât să dublați acest cod în mod inutil, creați o funcție ajutătoare care va fi apelată de ambele funcții, urmând pașii de mai jos. Parametrii pentru noua funcție sunt descriși mai târziu în această secțiune.

Crearea unei funcții ajutătoare

1. În cadrul paginii `ClassView`, selectați clasa `CMainFrame` și apoi alegeți opțiunea `Add Member Function` a meniului de context.
2. Introduceți `void` în caseta de editare `Function Type`.
3. Introduceți `CreateActivateView(CRuntimeClass *pNewView, UINT nID)` în caseta de editare `Function Declaration`.
4. Selectați butonul de opțiune `Private Access`. Efectuați un clic pe `OK`.

Tot ce mai rămâne de făcut este să scrieți codul. Ideea este ca utilizatorul să poată alege între reprezentarea de tip editor și cea artistică prin selectarea unei opțiuni de meniu corespunzătoare. Reprezentarea aleasă va fi afișată pe întreaga suprafață a ferestrei, iar cealaltă reprezentare va fi ascunsă. Opțiunile de meniu sunt activate sau dezactivate, în funcție de reprezentarea selectată curent. Editați funcția `CMapView::OnDraw` în concordanță cu listingul 20.4, prezentat mai devreme în acest capitol. Acesta face parte din exemplul anterior `SSplit` și este folosit doar pentru a ilustra diferența dintre reprezentări. Modificați funcțiile membru ale clasei `CMainFrame` așa cum o arată listingul 20.5.

Rețineți că va trebui să adăugați la începutul fișierului `MainFrm.cpp` directivele `#include` următoare:

```
#include "VPickView.h"
#include "ArtView.h"
```

De asemenea, trebuie să adăugați un antet de declarație pentru clasa document în fișierul `VPickView.h`, înainte de definiția de clasă. Această declarație este necesară din cauza ordinii în care sunt incluse acum fișierele antet. Adăugați linia următoare:

```
class CVPickDoc;
```

```
deasupra liniei:
```

```
class CVPickView : public CEditView
```

LISTINGUL 20.5 LST21_5.CPP – Implementarea opțiunilor de meniu pentru crearea și activarea reprezentărilor

```
1 void CMainFrame::OnShowEdit() — ❶
2 {
3     // ** Apelam functia ajutatoare, transmitand un pointer la
4     // ** clasa dinamica a reprezentarii si un identificator unic
5     CreateActivateView(RUNTIME_CLASS(CEditView), 1);
6 }
7
8 void CMainFrame::OnUpdateShowEdit(CCmdUI* pCmdUI)
9 {
10    // ** Activam/dezactivam optiunea de meniu in functie de
11    // ** clasa reprezentare activa
12    pCmdUI->Enable(
13        !GetActiveView()->IsKindOf(RUNTIME_CLASS(CEditView))); — ❷
14 }
15
16 void CMainFrame::OnShowArt() — ❸
17 {
18    // ** Apelam functia ajutatoare, transmitand un pointer la
19    // ** clasa dinamica a reprezentarii si un identificator unic
20    CreateActivateView(RUNTIME_CLASS(CartView), 1);
21 }
22
23 void CMainFrame::OnUpdateShowArt(CCmdUI* pCmdUI)
24 {
25    // ** Activam/dezactivam optiunea de meniu in functie de
26    // ** clasa reprezentare activa
27    pCmdUI->Enable(
28        !GetActiveView()->IsKindOf(RUNTIME_CLASS(CartView))); — ❹
29 }
30
31 void CMainFrame::CreateActivateView(
32     CRuntimeClass *pNewViewClass,
33     UINT nID)
34 {
35    // ** Obtinem un pointer la reprezentarea activa
36    CView* pOldView = GetActiveView();
37    CView* pNewView = NULL;
38
39    // ** Obtinem un pointer la documentul activ, dupa care
```

```
40    // ** parcurgem lista reprezentarilor asociate in cautarea
41    // ** unui obiect reprezentare a carui clasa dinamica este
42    // ** aceeași cu cea transmisa functiei
43    CDocument* pDoc = GetActiveDocument();
44    POSITION pos = pDoc->GetFirstViewPosition();
45    while(pos && !pNewView)
46    {
47        CView* pView = pDoc->GetNextView(pos); — ❺
48        if(pView->IsKindOf(pNewViewClass))
49            pNewView = pView;
50    }
51    // ** Verificam daca am gasit un obiect.
52    // ** Daca nu, construim, cream si initializam unul.
53    if (pNewView == NULL)
54    {
55        // ** Initializam o variabila CCreateContext cu un pointer
56        // ** la document
57        CCreateContext context;
58        context.m_pCurrentDoc = pDoc;
59
60        // ** Construim obiectul reprezentare pe baza
61        // ** clasei dinamice si cream fereastra asociata. — ❻
62        pNewView = (CView*)pNewViewClass->CreateObject();
63        pNewView->Create(NULL, NULL, 0,
64                        CFrameWnd::rectDefault,
65                        this, nID, &context); — ❼
66        pNewView->OnInitialUpdate();
67    }
68    // ** Noua reprezentare devine cea activa si fereastra sa este
69    // ** afisata. Ascundem fereastra celeilalte reprezentari.
70    SetActiveView(pNewView);
71    pNewView->ShowWindow(SW_SHOW);
72    pOldView->ShowWindow(SW_HIDE); — ❽
73
74    // ** Inversam identificatorii celor doua ferestre, pentru ca
75    // ** identificatorul ferestrei reprezentarii active trebuie
76    // ** sa fie AFX_IDW_PANE_FIRST.
77    pOldView->SetDlgCtrlID(pNewView->GetDlgCtrlID()); — ❾
78    pNewView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
79
80    // ** Recalculam pozitiile barelor de controale si dimensiunea
81    // ** ferestrei reprezentarii.
82    RecalcLayout();
83 }
```

❶ Funcție de tratare a comenzii de meniu.

- ② Dezactivăm opțiunea de meniu dacă CEditView este activă; altfel, o activăm.
- ③ Funcție de tratare a comenzii de meniu.
- ④ Dezactivăm opțiunea de meniu dacă CArtView este activă; altfel, o activăm.
- ⑤ Parcurge reprezentările asociate documentului în căutarea unei anumite clase reprezentare.
- ⑥ Construirea dinamică a unui obiect de tip dinamic.
- ⑦ Creăm reprezentarea și apelăm OnInitialUpdate.
- ⑧ Afișăm reprezentarea și o ascundem pe cealaltă.
- ⑨ Schimbăm între ei identificatorii de fereastră.

Observați că cele două funcții de tratare a meniurilor, OnShowEdit (linia 1) și OnShowArt (linia 16), apelează imediat funcția ajutoare CreateActivateView (prezentată începând cu linia 31). În aceste apeluri sunt transmiși doi parametri, și anume un pointer la informațiile dinamice asociate clasei reprezentare și un identificator unic. Primul parametru, m_pNewViewClass, este folosit pentru a deosebi tipul reprezentării selectate. Cel de-al doilea parametru este utilizat la identificarea ferestrei reprezentării.

Funcțiile OnUpdateShowEdit (linia 8) și OnUpdateShowArt (linia 23) sunt apelate de fiecare dată când se afișează elementele de meniu. Ele verifică clasa de care aparține reprezentarea activă, folosind funcția CObject::IsKindOf. Dacă clasa reprezentării active este aceeași cu clasa transmisă funcției IsKindOf, funcția CCmdUI::Enable primește valoarea FALSE și dezactivează opțiunea. Aceasta înseamnă că opțiunea corespunzătoare reprezentării ascunse este întotdeauna activată, în timp ce opțiunea care corespunde reprezentării vizibile este întotdeauna dezactivată.

Funcția CreateActivateView, care începe în linia 31, apelează mai întâi GetActiveView reținând în pOldView pointerul întors, acesta indicând reprezentarea care urmează să fie înlocuită. Următoarea sarcină este să determine dacă există o reprezentare de tip pNewViewClass asociată cu documentul activ. Prin intermediul ansamblului de funcții GetFirstViewPosition și GetNextView (liniile de la 44 la 50) se parcurg toate reprezentările asociate documentului. Dacă se găsește o reprezentare de tip pNewViewClass, în pNewView este înscrisă adresa acesteia, în linia 49. Testul if din linia 53 verifică dacă reprezentarea a fost găsită; dacă da, trebuie activată din nou, iar dacă nu, trebuie construită și activată.

Macrodefiniția ASSERT_KINDOF()

Pe lângă funcția IsKindOf(), orice obiect al unei clase derivate din CObject poate fi transmis macrodefiniției ASSERT_KINDOF(). Parametrii acesteia sunt un nume de clasă și un pointer la un obiect. Macrodefiniția verifică dacă obiectul aparține de clasa specificată; dacă nu, va fi afișată caseta de dialog MFC Assert. Macrodefiniția are efect doar în versiunea pentru depanare a unui proiect, fiind ignorată în versiunea finală.

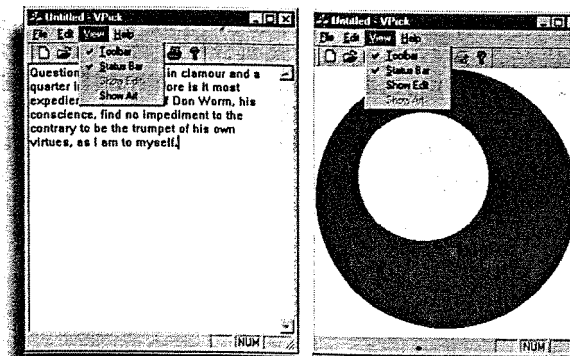
Construcția și crearea unui nou obiect reprezentare se petrece între liniile 57 și 67. Variabila context de tip CCreateContext este o structură care se inițializează cu un pointer către document. Structura este transmisă ulterior funcției Create, în linia 65, cu scopul de a stabili asocierea document/reprezentare. Obiectul reprezentare este instanțiat în linia 62; în aceste circumstanțe, pNewViewClass poate fi un obiect CEditView sau CArtView. Pentru că aceste clase permit crearea dinamică, funcția CreateObject va apela constructorul adecvat și va întoarce un pointer la noul obiect. Apelul Create din linia 63 creează fereastra reprezentării și o atașează la obiectul derivat din CView. În momentul în care reprezentarea este creată, se apelează funcția virtuală OnInitialUpdate(), acordându-i astfel posibilitatea de a se inițializa.

Indiferent dacă a fost nevoie de construirea sa, reprezentarea selectată este transmisă în apelul SetActiveView din linia 70, ceea ce duce la activarea sa și preluarea focusului. Rețineți că la activarea și dezactivarea reprezentărilor, acestea primesc un mesaj OnActivateView împreună cu un parametru bActivate de tip BOOL care indică starea. Linia 71 afișează reprezentarea selectată, iar linia 72 ascunde reprezentarea înlocuită.

Reprezentarea activă trebuie să aibă întotdeauna identificatorul AFX_IDW_PANE_FIRST, deoarece acesta este folosit explicit în RecalcLayout, dar nu poate exista decât o singură fereastră cu acest identificator. Așa stând lucrurile, linia 76 înlocuiește identificatorul ferestrei pOldView cu cel al ferestrei pNewView, stabilindu-l pe acesta din urmă la AFX_IDW_PANE_FIRST în linia 77. Apelul funcției RecalcLayout din linia 80 are ca efect dimensionarea ferestrei reprezentării și poziționarea barelor de instrumente și de controale pe suprafața ferestrei cadru.

Asamblați și rulați aplicația. Încercați să selectați fiecare dintre reprezentări. Figura 20.6 înfățișează aplicația VPick.

FIGURA 20.6
Exemplul VPick.



VEDEȚI...

- Pentru mai multe detalii privind meniurile, vedeți pagina 270.
- Pentru mai multe informații despre clasele MDI, vedeți pagina 477.
- Pentru amănunte privind informațiile de clasă dinamice, vedeți pagina 532.

Dezvoltarea aplicațiilor cu documente multiple

Crearea unei aplicații cu interfață de tip document multiplu

Accesarea obiectelor aplicație, document, reprezentare și cadru

Dezvoltarea unei aplicații cu ajutorul arhitecturii MFC
Document/Reprezentare

Multe dintre cele mai populare și mai folosite aplicații din ziua de astăzi se bazează pe arhitectura *Document/Reprezentare* sau pe variante ale sale. Această arhitectură aduce o structură modulară a programului, în care codul care administrează datele este separat de codul responsabil de interfața cu utilizatorul. Datele sunt gestionate în cadrul documentului, iar interfața cu utilizatorul este implementată în cadrul reprezentării. De regulă, documentul corespunde direct unui fișier de pe disc.

Infrastructura MFC oferă două tipuri de aplicații *Document/Reprezentare*: *SDI* (*interfață document singular*) și *MDI* (*interfață document multiplu*).

Aplicațiile *SDI* sunt prezentate în capitolul 12, „Utilizarea documentelor, a reprezentărilor și a cadrelor”. Acest capitol discută aspectele de bază ale arhitecturii *Document/Reprezentare*, în linii mari aceleași pentru aplicațiile *SDI* și *MDI*. În cazul în care nu sunteți familiar cu paradigma *Document/Reprezentare*, consultați în paralel capitolul 12. Am utilizat în mod deliberat același cod în ambele capitole, înlesnind astfel comparația între cele două tipuri de aplicații.

Diferența dintre aplicațiile *document singular* și cele de tip *document multiplu* este explicată destul de bine chiar din denumire. În aplicațiile *SDI*, la un moment dat nu poate fi deschis decât un singur document. În aplicațiile *MDI* pot fi deschise simultan unul sau mai multe documente, fiecare având o fereastră reprezentare proprie care realizează interfața cu utilizatorul.

VEDEȚI...

➤ Pentru mai multe informații privind aplicațiile *SDI*, vedeți pagina 246.

Crearea unei aplicații cu interfață de tip document multiplu

S-a discutat mult dacă aplicațiile ar trebui create sub formă *SDI* sau *MDI*. O serie de opinenți susțin de mai mulți ani ideea că programele ar trebui să adopte o abordare strict orientată pe document, considerând că *SDI* ar trebui să devină standard. Dar aplicațiile sunt create pentru utilizatori și aceștia arată, prin volumul achizițiilor, ce doresc de la produsele cumpărate. Aplicațiile *MDI*, precum *MS Word* sau *Excel*, se vând mai bine decât majoritatea celorlalte aplicații. Pentru anumite tipuri de pachete software, capacitatea de a vedea și gestiona mai multe documente (sau fișiere) în cadrul aceleiași instanțe a aplicației este de preferat față de lansarea programului de mai multe ori.

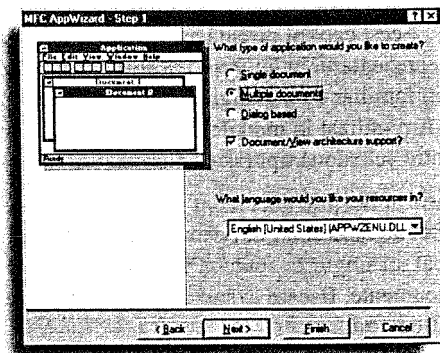
Generarea unui proiect *MDI* nu ține decât de selectarea unor opțiuni adecvate în *AppWizard*. Trebuie să știți, însă, că după crearea proiectului nu mai este deloc ușor să schimbați între *SDI* și *MDI*. Depinde de cât de departe ați ajuns cu dezvoltarea, dar s-a putea să fie mai ușor să o luați de la început cu un nou proiect. Vă recomand să vă gândiți bine, înainte de a începe, ce tip de interfață se potrivește cel mai bine nevoilor aplicației.

În această secțiune veți crea o aplicație *MDI* pe care o vom folosi pentru a studia clasele generate și modul în care funcționează arhitectura *document/reprezentare*. Creați un nou proiect numit *MDICoin*, urmând pașii de mai jos.

Crearea unei aplicații MDI cu AppWizard

1. Selectați **New** din meniul **File**; apoi selectați **MFC AppWizard (exe)** din pagina **Project** a casetei de dialog **New Project**.
2. Introduceți numele proiectului (**MDICoin**, pentru exemplul nostru) și apoi efectuați clic pe **OK**.
3. În caseta de dialog **MFC AppWizard Step 1**, selectați butonul de opțiune **Multiple Documents** și asigurați-vă că este validată caseta **Document/View Architecture Support?** (vedeți figura 21.1). Efectuați un clic pe **Next**.

FIGURA 21.1
Pasul 1 din AppWizard



4. Selectați butonul de opțiune **None** din caseta de dialog corespunzătoare pasului 2. Opțiunile prezente aici permit alegerea unor diferite tipuri de suport pentru baze de date, ceea ce nu este necesar în cazul nostru. Efectuați un clic pe **Next**.
5. În caseta de dialog corespunzătoare pasului 3, selectați butonul de opțiune **None** și validați caseta **ActiveX Controls**. Celelalte opțiuni privesc facilități de automatizare OLE de care nu avem nevoie aici. Efectuați un clic pe **Next**.
6. În caseta de dialog corespunzătoare pasului 4, validați următoarele opțiuni: **Docking Toolbar**, **Initial Status Bar**, **Printing and Print Preview** și **3D Controls**. Selectați butonul de opțiune **Normal** pentru a preciza stilul barei cu instrumente. (Vedeți figură 21.2 pentru opțiunile corecte.) Efectuați un clic pe **Next**.
7. În caseta de dialog corespunzătoare pasului 5, selectați **MFC Standard** ca tip de proiect, selectați **Yes** pentru a genera comentarii în codul sursă și selectați **As a Shared DLL** pentru a specifica modul de utilizare a bibliotecii MFC. Efectuați un clic pe **Next**.
8. În caseta de dialog corespunzătoare pasului 6, efectuați un clic pe **Finish**. Va fi afișată caseta de dialog **New Project Information**, ilustrată în figura 21.3. Aici se află câteva informații utile privind clasele generate și numele fișierelor sursă. Puteți, de asemenea, să confirmați facilitățile selectate. Efectuați un clic pe **OK**. Noul proiect va fi creat acum și deschis în Developer Studio.

FIGURA 21.2
Pasul 4 din AppWizard.

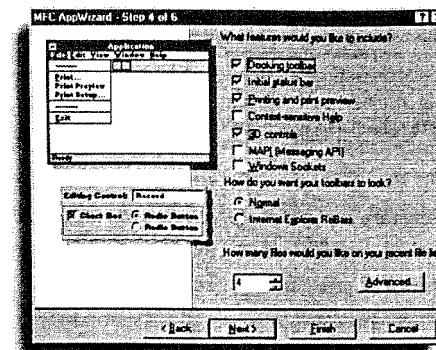
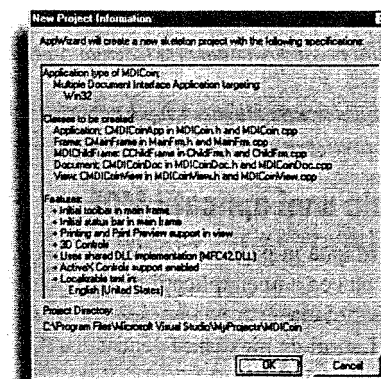


FIGURA 21.3
Casetă de dialog New Project Information.



Veți implementa o aplicație care afișează un teanc de monede. Opțiunile de meniu vă vor permite să depuneți sau să extrageți monede din teanc. Datele constă în numărul de monede, număr reținut în clasa document și accesat de clasa reprezentare în scopul afișării teancului. Exemplul **MDICoin** are la bază programul **SDICoin** din capitolul 12, codul fiind foarte asemănător. Dar **MDICoin** va permite deschiderea simultană a mai multor ferestre Document/Reprezentare. Vom dezvolta exemplul pentru a gestiona un nou tip de document, care afișează nu monede, ci bancnote. Dacă o aplicație cu interfață document singular are o singură clasă derivată din **CDocument**, o aplicație cu interfață document multiplu poate avea oricâte astfel de clase.

Fiecare clasă document este asociată cu o clasă reprezentare. Clasa reprezentare poate fi utilizată de mai multe documente diferite sau poate aparține unui singur tip de document. Clasa de bază pentru reprezentare este selectată în pasul 6 din AppWizard; opțiunile disponibile sunt **CView**, **CScrollView**, **CListView**, **CTreeView**, **CEditView**, **CRichEditView**, **CFormView**, **CRecordView** și **CHtmlView**. Exemplul **MDICoin** folosește clasa **CView** propusă implicit.

Documentele încapsulează datele aplicațiilor

Clasa document este derivată din `CDocument` și variabilele sale membru conțin datele aplicației. Funcțiile membru ale clasei document sunt cele care permit manipularea datelor. De exemplu, funcția `Serialize()` este folosită pentru citirea și scrierea datelor pe disc.

VEDEȚI...

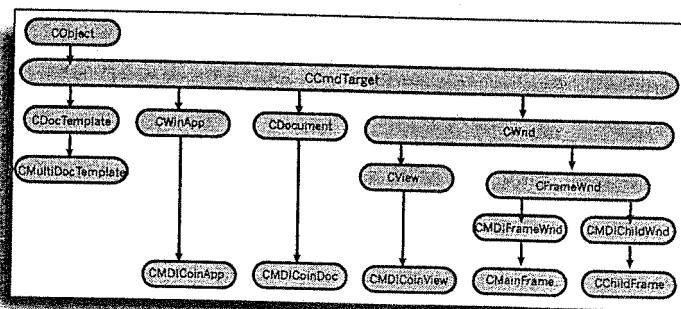
- Pentru o prezentare a claselor reprezentare din MFC, vedeți tabelul 12.1 de la pagina 249.
- Pentru informații despre clasa `CListView`, vedeți pagina 427.
- Pentru informații despre clasa `CTreeView`, vedeți pagina 441.
- Pentru informații despre clasa `CRichEditView`, vedeți pagina 449.
- Pentru informații despre clasa `CHTMLView`, vedeți pagina 454.
- Pentru informații despre clasa `CScrollView`, vedeți pagina 415.
- Pentru informații despre clasele `CFormView` și `CRecordView`, vedeți pagina 577.
- Pentru informații privind manipularea fișierelor în cadrul arhitecturii Document/Reprezentare, vedeți pagina 532.
- Pentru informații despre adăugarea suportului pentru baze de date, vedeți pagina 568.
- Pentru informații despre automatizarea OLE, vedeți pagina 589.

Despre clasele unei aplicații MDI

Ierarhia de clase afișată în pagina ClassView a unei aplicații de tip document multiplu este aproape identică cu cea a unei aplicații de tip document singular. Singura diferență vizibilă este prezența unei noi clase, `CChildFrame`. În spatele aparențelor se ascund, însă, mai multe deosebiri. Lanțul de moștenire pentru clasele generate de AppWizard este ilustrat în figura 21.4.

FIGURA 21.4

Ierarhia de clase pentru o aplicație MDI.



Structura de ansamblu a aplicațiilor SDI și MDI este aproape aceeași. Clasa `CMDICoinApp`, derivată din `CWinApp`, se ocupă de inițializarea aplicației. Clasa de bază, `CWinApp`, întreține asocierea dintre clasele document, reprezentare și cadru. Primește, totodată, mesaje generate de Windows și le trimite către fereastra adecvată.

Clasa `CMDICoinDoc`, derivată din `CDocument`, are rolul de container pentru datele aplicației, fiind responsabilă de serializarea datelor într-o formă de stocare permanentă, cum ar fi un

fișier pe disc. Spre deosebire de aplicațiile SDI, în care există o singură instanță a clasei document, într-o aplicație MDI pot exista simultan mai multe obiecte document. Fiecare dintre acestea răspunde de gestionarea datelor corespunzătoare unui fișier. Crearea unui obiect document duce întotdeauna la crearea unei noi ferestre reprezentare, asocierea acestuia cu documentul și crearea unei noi ferestre cadru care va conține reprezentarea. Pentru că la un moment dat pot exista mai multe instanțe de document, infrastructura MDI introduce conceptul de *document activ*. Un document devine activ în momentul în care utilizatorul interacționează cu una din reprezentările asociate acestuia.

Documentul și reprezentarea active

O fereastră reprezentare devine activă în momentul în care utilizatorul începe să interacționeze cu aceasta. La schimbarea reprezentării active este apelată funcția `OnActivate()`. Această funcție este apelată mai întâi pentru fereastra dezactivată și apoi pentru fereastra care se activează. Noua stare a ferestrei este dată de un parametru care poate lua una dintre următoarele valori: `WA_INACTIVE`, `WA_ACTIVE` sau `WA_CLICKACTIVE`. Obiectul document asociat cu reprezentarea activă se numește document activ.

Clasa `CMDICoinView` este responsabilă de generarea pentru utilizator a reprezentării vizuale a datelor documentului, permițând interacțiunea acestuia cu datele. În exemplul MDICoin, fiecare reprezentare va desena un teanc de monede. O reprezentare este asociată cu o singură instanță a unui document, în timp ce un document poate avea asociate mai multe obiecte reprezentare. Clasele reprezentare accesează datele necesare prin intermediul funcțiilor membru publice ale clasei document.

Rolul clasei `CMainFrame` este să furnizeze fereastra aplicației. Implementarea ferestrei cadru diferă între aplicațiile SDI și MDI. Aplicațiile MDI derivă `CMainFrame` din `CMDIFrameWnd`, care creează o fereastră cadru distinctă pentru aplicație (numită, de regulă, fereastră cadru principală) și câte o fereastră pentru fiecare reprezentare. Fereastra cadru principală conține bara(-ele) cu instrumente și bara de stare și este responsabilă de propria administrare. De exemplu, funcția `CFrameWnd::RecalcLayout` are ca efect poziționarea barei cu instrumente și a barei de sarcini și dimensionarea ferestrei reprezentare.

Fiecare fereastră reprezentare se află într-o fereastră cadru proprie, implementată de clasa `CChildFrame`. Așa cum se vede în figura 21.4, `CChildFrame` este derivată din `CMDIChildWnd`. Pentru a putea furniza funcționalitatea suplimentară a arhitecturii MDI, această clasă folosește o fereastră specială oferită de sistemul Windows, `MDIClient`. La crearea unei ferestre cadru MDI se creează și o fereastră copil `MDIClient` a acesteia, iar ferestrele copil MDI sunt copii ai ferestrei `MDIClient`, ceea ce le face „nepoți” ai ferestrei cadru principale.

VEDEȚI...

- Pentru a afla mai multe despre diferențele dintre clasele MDI și SDI, vedeți pagina 250.

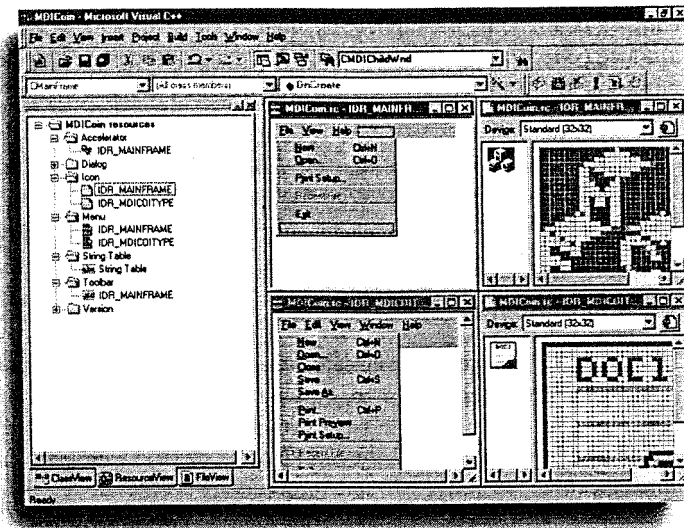
Elementele vizuale ale unei aplicații MDI

În capitolul 12 ați văzut că AppWizard generează resursele necesare pentru o aplicație SDI, cum ar fi meniul, bara cu instrumente și așa mai departe, toate acestea având același identificator, `IDR_MAINFRAME`. În cazul unei aplicații MDI, AppWizard generează câteva

resurse adiționale. Acestea sunt necesare din două motive. În primul rând, pentru că o aplicație MDI poate gestiona mai multe tipuri de documente, iar în al doilea rând, pentru că o aplicație SDI conține întotdeauna un obiect document, ceea ce nu este valabil în cazul aplicațiilor MDI. Figura 21.5 înfățișează resursele din cadrul exemplului MDICoin.

FIGURA 21.5

Resursele unei aplicații MDI.



Pe lângă resursele `IDR_MAINFRAME`, AppWizard a creat o pictogramă, un meniu și intrări în tabela de șiruri corespunzătoare tipului de document, atribuindu-le tuturor identificatorul `IDR_MYCOINTYPE`. În acest fel aveți posibilitatea să modificați resurse în funcție de document.

Pictograma documentului este afișată în partea stângă a meniului în momentul în care fereastra reprezentare este maximizată, respectiv pe bara de titlu a ferestrei reprezentare în celelalte cazuri.

Resursa meniu asociată cu tipul de document intră automat în vigoare la activarea obiectului document. În cazul în care nu există nici un document activ, situație perfect posibilă în cazul aplicațiilor MDI, este activat meniul `IDR_MAINFRAME`. Figura 21.5 prezintă diferența dintre meniul **File** principal și meniul **File** al documentului. În MDI apare, de asemenea, un meniu **Window** care permite utilizatorului să ordoneze ferestrele reprezentare cu ajutorul opțiunilor **Tile** și **Cascade**. Acest meniu conține și o opțiune **New Window** care adaugă automat suportul necesar pentru a permite utilizatorului să vadă mai multe reprezentări, fiecare într-o fereastră cadru distinctă.

Meniuri specifice documentului

Fiecare tip de document din cadrul unei aplicații MDI are asociată o resursă meniu proprie, `IDR_`, care este afișată automat în momentul în care un obiect document corespunzător devine activ. Într-un program MDI este posibil să nu existe nici un document activ, caz în care este afișată resursa meniu `IDR_MAINFRAME`.

Există și o intrare în tabela de șiruri având identificatorul `IDR_MYCOINTYPE`, aceasta conținând informații despre șapte aspecte distincte ale documentului și fiind utilizată, printre altele, pentru dispunerea ferestrelor alăturat.

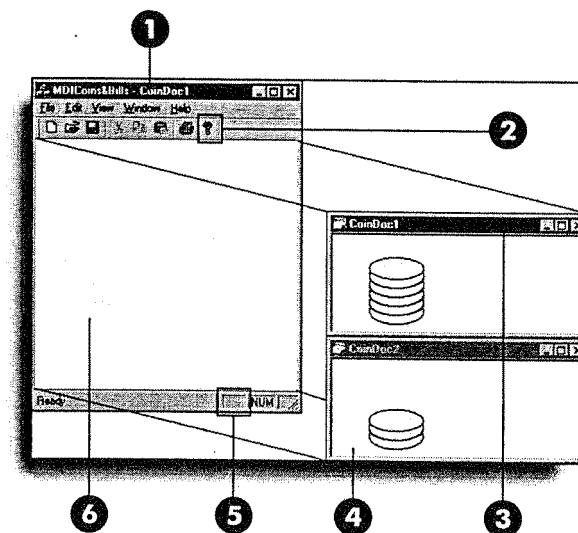
Figura 21.6 enumeră elementele vizuale ale unei aplicații MDI. Iată și explicațiile aferente:

- `CMainFrame` este derivată din `CMDIFrameWnd` și furnizează fereastra cadru principală a aplicației.
- `CToolBar` implementează barele cu instrumente ancorate de fereastra cadru principală.
- `CChildFrame` este derivată din `CMDIChildWnd` și implementează o fereastră cadru pentru fiecare reprezentare.
- `CMDICoinView` implementează reprezentarea ca și copil al unei ferestrei cadru MDI.
- `CStatusBar` implementează bara de stare atașată de fereastra cadru principală.
- Ferestrele reprezentare MDI se suprapun peste zona client a ferestrei cadru principale.

FIGURA 21.6

Elementele vizuale ale unei aplicații MDI.

- 1 `CMainFrame`
- 2 `CToolBar`
- 3 `CChildFrame`
- 4 `CMDICoinView`
- 5 `CStatusBar`
- 6 Ferestrele reprezentare MDI



VEDEȚI...

- Pentru informații despre meniuri, vedeți pagina 270.
- Pentru informații despre barele cu instrumente, vedeți pagina 292.

Despre machetele de document MDI

La baza arhitecturii Document/Reprezentare se află machetele de document. Folosite și în aplicațiile SDI, acestea joacă un rol mult mai important la dezvoltarea aplicațiilor MDI. O machetă de document face legătura dintre document, reprezentare și fereastra cadru, dat fiind un tip particular de document. Într-un proiect cu interfață document singular nu este

nevoie, de obicei, să interveniți asupra codului generat de AppWizard care se ocupă de machetele de document. Într-un proiect cu interfață document multiplu, însă, ar putea fi necesar să creați noi tipuri de documente și să asociați acestora diferite clase cadru și reprezentare, ceea ce înseamnă că trebuie să implementați cod necesar pentru machetele de document.

Din fericire, adăugarea de cod pentru noi tipuri de document nu este dificilă, mai ales dacă țineți cont de faptul că AppWizard a generat deja codul corespunzător pentru un prim document, cu reprezentarea și cadrul asociate, oferindu-vă un exemplu relativ ușor de urmat. Dacă priviți figura 21.4, care prezintă ierarhia de clase dintr-o aplicație MDI, veți vedea clasa `CMultiDocTemplate`, care este clasa machetă de document pentru aplicațiile cu interfață document multiplu.

Cam pe la jumătatea funcției `CMDICoinApp::InitInstance` (care este apelată de infrastructura MFC pentru inițializarea aplicației) veți vedea codul următor:

```
1 // Register the application's document
  templates. Document templates
2 // serve as the connection between documents,
  frame windows, and views.
3 CMultiDocTemplate* pDocTemplate;
4 pDocTemplate = new CMultiDocTemplate( — ❶
5     IDR_MDICOITYPE,
6     RUNTIME_CLASS(CMDICoinDoc),
7     RUNTIME_CLASS(CChildFrame),
8     RUNTIME_CLASS(CMDICoinView));
9 AddDocTemplate(pDocTemplate); — ❷
10
11 // Create main MDI Frame window
12 CMainFrame* pMainFrame = new CMainFrame; — ❸
13 if (!pMainFrame->LoadFrame(IDR_MAINFRAME)) — ❹
14     return FALSE;
15 m_pMainWnd = pMainFrame;
```

❶ Se instanțiază un obiect machetă de document multiplu, fiind transmise identificatorul resurselor asociate și clasele dinamice document, reprezentare și cadru.

❷ Înregistrează macheta de document nou creată în clasa aplicației.

❸ Instanțiază fereastra cadru a aplicației.

❹ Indică ferestrei cadru principale setul de resurse care trebuie încărcate (printre care meniul sau bara cu instrumente).

Se creează un obiect `CMultiDocTemplate`, fiind transmiși patru parametri (linia 4). Primul parametru este identificatorul de resurse, `IDR_MDICOITYPE` (linia 5). Acesta corespunde la trei resurse distincte, toate asociate cu documentul `CMDICoinDoc`: o pictogramă, un meniu și o intrare în tabela de șiruri. Următorii trei parametri sunt toți pointeri la informații de

clasă dinamice pentru clasele document, cadru și, respectiv, reprezentare (liniile 6, 7 și 8). Acești pointeri sunt generați cu ajutorul macrodefiniției `RUNTIME_CLASS`. Așa ceva este posibil deoarece AppWizard a inclus suport pentru crearea dinamică a acestor clase, incorporând macrodefinițiile `DECLARE_DYNCREATE` și `IMPLEMENT_DYNCREATE`.

Între machetele de document SDI și MDI există o serie de diferențe. Clasele `CSingleDocTemplate` și `CMultiDocTemplate` sunt derivate ambele din aceeași clasă de bază, `CDocTemplate`. Dar în timp ce `CSingleDocTemplate` conține un singur obiect document, `CMultiDocTemplate` păstrează o listă de pointeri la documente (în acest caz, obiecte `CMDICoinDoc`).

Obiectele document, reprezentare și cadru propriu-zise nu sunt create acum. Codul inițializează obiectul `CMultiDocTemplate` cu informațiile necesare pentru încărcarea resurselor și crearea la cerere a documentelor, reprezentărilor și cadrelor.

Clasa aplicație păstrează o listă de machete de document. Apelul funcției `AddDocTemplate` din linia 9 adaugă obiectul machetă de document în această listă. Puteți să apelați `AddDocTemplate` ori de câte ori doriți pentru a înregistra și alte machete de document. În acest fel, o singură aplicație poate să gestioneze mai multe asocieri de documente, reprezentări și cadre.

Caseta de dialog pentru crearea unui nou document

Dacă în clasa aplicației sunt înregistrate mai multe tipuri de document, selectarea opțiunii **New** din meniul **File** va duce la afișarea unei casete de dialog care solicită utilizatorul să aleagă tipul de document ce va fi creat. Descrierea afișată în caseta de dialog este preluată din intrarea corespunzătoare documentului din tabela de șiruri.

Clasa `CWinApp` păstrează obiectul `CMultiDocTemplate` până când este distrusă, ceea ce se întâmplă doar la încheierea aplicației. În procesul de distrugere, clasa `CWinApp` eliberează orice memorie alocată pentru machetele de document, cu condiția ca acestea să fi fost adăugate în lista aplicației prin apelul funcției `AddDocTemplate`. Machetele de document folosesc cantități infime de memorie, așa că existența lor pe întreaga durată de viață a aplicației nu ridică probleme.

În linia 12, fereastra cadru principală este creată prin intermediul constructorului `CMainFrame`. Funcția `LoadFrame` din linia 13 primește identificatorul `IDR_MAINFRAME` al resurselor care trebuie încărcate și atașate ferestrei cadru. În apelul `LoadFrame` puteți transmite un al doilea parametru, opțional, care specifică indicatori de stil pentru fereastră. Implicit, stilurile ferestrei sunt `WS_OVERLAPPEDWINDOW` și `FWS_ADDTOTITLE`. Stilul `WS_OVERLAPPEDWINDOW` este o combinație între opțiunile de stil prezentate în tabelul 21.1. Indicatorul `FWS_ADDTOTITLE` este specific MFC și face ca numele documentului activ să fie afișat pe bara de titlu. În mod normal se afișează mai întâi numele aplicației, urmat de numele documentului activ. Această ordine poate fi inversată prin adăugarea indicatorului de stil `FWS_PREFIXTITLE`.

TABELUL 21.1 Stilurile pentru fereastra cadru principală incluse în WS_OVERLAPPEDWINDOW

Stil	Descriere
WS_OVERLAPPED	Creează o fereastră suprapusă
WS_CAPTION	Dotează fereastra cu o bară de titlu
WS_SYSMENU	Adaugă pe bara de titlu un meniu sistem implicit
WS_MINIMIZEBOX	Adaugă pe bara de titlu un buton de minimizare
WS_MAXIMIZEBOX	Adaugă pe bara de titlu un buton de maximizare
WS_THICKFRAME	Permite redimensionarea ferestrei

VEDEȚI...

► Pentru a afla mai multe despre informațiile de clasă dinamice, vedeți pagina 532.

Secvența de creare a documentului, reprezentării și a cadrului MDI

Așa cum am menționat în secțiunea anterioară, inițializarea obiectului machetă de document nu duce la construirea obiectelor document, reprezentare sau fereastră copil MDI. Machetele de document nu fac decât să rețină informații de clasă dinamice pentru fiecare mulțime document/reprezentare/cadru. Aceste informații dinamice sunt folosite la crearea obiectelor și păstrarea pointerilor corespunzători în obiectul aplicație doar atunci când utilizatorul decide să creeze un nou document sau să deschidă unul deja existent.

După crearea și inițializarea obiectului CMultiDocTemplate și după crearea cu succes a ferestrei cadru principale, în funcția CWinApp::InitInstance urmează codul de mai jos:

```

1 // Parse command line for standard shell command, DDE,
  // file open
2 CCommandLineInfo cmdInfo;
3 ParseCommandLine(cmdInfo);
4 // Dispatch commands specified on the command line
5 if (!ProcessShellCommand(cmdInfo))
6     return FALSE;
7 // The one and only window has been initialized,
  // so show and update it.
8 m_pMainWnd->ShowWindow(SW_SHOW);
9 m_pMainWnd->UpdateWindow();

```

Clasa CCommandLineInfo se ocupă de tratarea parametrilor trimiși aplicației din linia de comandă. Opțiunile transmise implicit în linia de comandă determină crearea unui nou document la lansarea aplicației. Construcția efectivă a documentului, a reprezentării și a ferestrei cadru MDI se petrece în culise, în cadrul funcției

CWinApp::ProcessShellCommand apelată în linia 5. Listingul 21.1 oferă o imagine de ansamblu asupra secvenței de creare. Vă atrag atenția că întreg codul din listingul 21.1 este parafrizat și a fost condensat pentru a scoate în evidență procesul de creare.

LISTINGUL 21.1 LST22_5.CPP – Secvența de creare a documentului, reprezentării și a ferestrei cadru MDI

```

1 BOOL CWinApp::ProcessShellCommand(CCommandLineInfo& rCmdInfo) — ❶
2 {
3     if(rCmdInfo.NewFile)
4     {
5         OnFileNew();
6     }
7     if(rCmdInfo.OpenFile)
8     {
9         OpenDocumentFile(rCmdInfo.strFileName);
10    }
11    ...
12 }
13 void CWinApp::OnFileNew() — ❷
14 {
15     m_pDocManager->OnFileNew();
16 }
17 void CDocManager::OnFileNew()
18 {
19     pTemplate = m_TemplateList.GetHead();
20     if(m_templateList.GetCount() > 1) — ❸
21     {
22         // more than one document Template
23         // bring up dialog prompting user
24         pTemplate = dlg.m_pSelectedTemplate; — ❹
25     }
26     pTemplate->OpenDocumentFile(NULL);
27 }
28 CMultiDocTemplate::OpenDocumentFile(LPCTSTR lpszPathName)
29 {
30     pDocument = CreateNewDocument();
31     pFrame = CreateNewFrame(pDocument); — ❺
32     ...
33     if(lpszPathName == NULL)
34     {
35         SetDefaultTitle(pDocument);
36         pDocument->OnNewDocument();
37     }
38     else
39     {
40         pDocument->OnOpenDocument(); — ❻
41         InitialUpdateFrame(pFrame, pDocument);
42     } — ❼

```

❶ Apelată de funcția InitInstance.

- 2 Funcție de tratare și pentru opțiunea **New** din meniul **File**.
- 3 `pTemplate` este un pointer la obiectul `CMultiDocTemplate`.
- 4 Dacă au fost înregistrate mai multe tipuri de document, o casetă de dialog solicită utilizatorul să selecteze unul dintre acestea.
- 5 Obiectele document și cadru sunt construite folosind informațiile de clasă dinamice din machetă. `CreateNewFrame` construiește și reprezentarea.
- 6 `OnNewDocument` și `OnOpenDocument` sunt funcții virtuale care sunt supradefinite pentru a se efectua inițializarea documentului.
- 7 Se activează reprezentarea și se trimite un mesaj `WM_INITIALUPDATE`, care duce la apelarea funcției `CView::OnInitialUpdate`.

Diferența dintre secvențele de creare pentru structurile SDI și MDI constă în faptul că modelul MDI creează un nou obiect document de fiecare dată când se apelează.

`OpenDocumentFile` (linia 26), în timp ce modelul SDI creează un singur obiect document și apoi îl refolosește. Funcția `CreateNewDocument`, apelată în linia 30, folosește informațiile de clasă dinamice din cadrul obiectului machetă pentru a construi obiectul document specific aplicației. De asemenea, funcția adaugă documentul la lista documentelor curente. După crearea documentului se creează și cadrul și reprezentarea, în acest scop fiind apelată o altă funcție a obiectului machetă, `CreateNewFrame`.

Funcția `CreateNewFrame`, apelată în linia 31, folosește informațiile de clasă dinamice din cadrul obiectului machetă pentru a construi obiectele fereastră cadru și reprezentare. Odată creat cadrul, este apelată funcția `CFrameWnd::OnCreate`, fiind transmis identificatorul de resurse specificat în constructorul obiectului machetă. În cazul aplicațiilor MDI, acest identificator depinde de document. De pildă, exemplul `MDICoin` transmite identificatorul `IDR_MDICOIN`. `LoadFrame` încarcă fiecare resursă (meniu, bară cu instrumente, pictogramă, tabelă de acceleratori și șir de caractere) și o atașează la fereastra cadru. Rețineți că dacă încărcarea unei resurse eșuează, funcția `LoadFrame` va eșua la rândul său, generând mesajul de avertisment: `Warning: CDocTemplate couldn't create a frame.`

În finalul secvenței de creare se apelează funcția `InitialUpdateFrame` a obiectului machetă (linia 40). Aceasta activează reprezentarea nou creată și apoi trimite mesajul `WM_INITIALUPDATE` către copii ferestrei cadru, printre aceștia aflându-se fereastra reprezentare. La primirea mesajului este apelată funcția virtuală `CView::OnInitialUpdate`. Se obișnuiește supradefinirea acestei funcții pentru efectuarea de inițializări specifice reprezentării. Rețineți că de obicei este necesar să apelezi și versiunea `OnInitialUpdate` din clasa de bază.

VEDEȚI...

- Pentru informații despre parametrii din linia de comandă, vedeți pagina 255.

Accesarea obiectelor document/reprezentare

Atunci când folosiți paradigma Document/Reprezentare veți avea nevoie să accesați diferitele obiecte unul dintr-altul. Cele două componente care interacționează cel mai mult sunt documentul și reprezentarea asociată, dar apar multe situații care implică interacțiuni între alte clase. Spre exemplu, ați putea avea în clasa aplicație variabile membru pe care doriți să le accesați dintr-o clasă reprezentare. Tabelul 21.2 prezintă funcțiile oferite de biblioteca MFC în scopul accesării diferitelor obiecte ale unei aplicații MDI.

TABELUL 21.2 Funcții de acces MDI

Clasă	Funcție	Rezultat
Global	<code>AfxGetApp</code>	Un pointer la obiectul <code>CWinApp</code> .
Global	<code>AfxGetMainWnd</code>	Un pointer la obiectul fereastră cadru principală (<code>CWnd</code>).
<code>CMDIFrameWnd</code>	<code>MDIGetActive</code>	Un pointer la obiectul fereastră copil MDI activ (<code>CMDIChildWnd</code>).
<code>CWnd</code>	<code>GetParentFrame</code>	Un pointer la obiectul fereastră cadru părinte (<code>CFrameWnd</code>).
<code>CFrameWnd</code>	<code>GetActiveView</code>	Un pointer la obiectul reprezentare activ curent (<code>CView</code>).
<code>CFrameWnd</code>	<code>GetActiveDocument</code>	Un pointer la obiectul document activ curent (<code>CDocument</code>).
<code>CView</code>	<code>GetDocument</code>	Un pointer la obiectul document (<code>CDocument</code>) asociat reprezentării.
<code>CDocument</code>	<code>GetFirstViewPosition</code>	Poziția primei reprezentări din lista reprezentărilor asociate cu documentul. Folosită pentru a parcurge reprezentările unui document.
<code>CDocument</code>	<code>GetNextView</code>	Un pointer la următorul obiect reprezentare (<code>CView</code>) din lista reprezentărilor asociate cu documentul.

`AfxGetApp` și `AfxGetMainWnd` sunt funcții globale și pot fi apelate de oriunde. Pointerii întorși de aceste două funcții pot fi convertiți fără probleme la clasele specifice aplicației, așa cum o arată exemplele următoare:

```
CMDICoinApp* pApp = (CMDICoinApp*)AfxGetApp();
CMainFrame* pFrame = (CMainFrame*)AfxGetMainWnd();
```

O reprezentare este asociată cu un singur document, așa că existența unei funcții `GetDocument` într-o clasă reprezentare este suficientă. Pointerul întors de această funcție este convertit automat la clasa document din aplicație prin codul generat de `AppWizard`.

Un document poate avea asociate mai multe reprezentări și pot exista mai multe obiecte document. Pentru a accesa documentul activ sau reprezentarea activă trebuie să identificați

mai întâi fereastra copil MDI activă, după care apelezi `GetActiveDocument` sau `GetActiveView`, așa cum se vede în exemplul de mai jos:

```
// Gasim fereastra copil MDI activa
CMDIChildWnd* pChild =
    ((CMDIFrameWnd*)AfxGetMainWnd())->MDIGetActive();
// Obținem documentul activ
CMDICoinDoc* pDoc =
    (CMDICoinDoc*)pChild->GetActiveDocument();
// Obținem reprezentarea activa
CMDICoinView* pView =
    (CMDICoinView*)pChild->GetActiveView();
```

Dacă trebuie să efectuați o operație pe unele sau pe toate reprezentările asociate cu un document anume, folosiți funcțiile `GetFirstViewPosition` și `GetNextView` ale clasei document pentru a parcurge toate reprezentările documentului, așa cum se ilustrează în exemplul care urmează:

```
void CmyDoc::DoTaskForAllViews()
{
    POSITION pos = GetFirstViewPosition();
    while (pos != NULL)
    {
        CMyView* pMyView = GetNextView(pos);
        pMyView->DoTask();
    }
}
```

O reprezentare corespunde unui singur document

Un obiect document poate avea mai multe reprezentări, dar o reprezentare este asociată cu un singur document.

Dezvoltarea unui exemplu de aplicație MDI

La începutul acestui capitol ați creat proiectul MDICoin. Până acum, scopul său a fost să ofere exemple de cod generat de AppWizard, exemple care să ilustreze modul în care funcționează infrastructura MDI. În secțiunea de față veți dezvolta acest proiect, într-o primă fază afișându-se un teanc de monede care pot fi adăugate sau înlăturate prin intermediul unor opțiuni de meniu. Apoi veți crea un al doilea tip de document care va afișa bancnote. Acest al doilea tip va presupune existența unor clase document și reprezentare specifice, precum și a unor resurse proprii. Va fi necesar, de asemenea, un obiect machetă de document care trebuie creat, inițializat și înregistrat în cadrul aplicației.

Adăugarea de variabile membru în document

În clasa document `CMDICoinDoc` este nevoie de o singură variabilă membru, care va reține numărul de monede aflate curent în teanc. Pentru ca clasa reprezentare `CMDICoinView` să

poată accesa această variabilă, va trebui să adăugați în clasa document o funcție care să furnizeze valoarea sa. Urmați pașii de mai jos pentru a adăuga codul necesar.

Implementarea stocării datelor documentului și a metodelor de acces

1. Selectați clasa `CMDICoinDoc` din cadrul paginii `ClassView`; apoi selectați **Add Member Variable** din meniul contextual. Va fi afișată caseta de dialog **Add Member Variable**.
2. Introduceți `int` în caseta de editare **Variable Type**. Introduceți `m_nCoins` în caseta de editare **Variable Name** și selectați butonul de opțiune **Protected Access**. Efectuați un clic pe **OK**.
3. Având clasa `CMDICoinDoc` încă selectată, alegeți opțiunea **Add Member Function** prezentă în meniul de context.
4. Introduceți `int` în caseta de editare **Function Type**. Introduceți `GetCoinCount` în caseta de editare **Function Declaration** și selectați butonul de opțiune **Public Access**. Introduceți în corpul funcției `GetCoinCount` următoarea linie de cod:

```
return m_nCoins;
```
5. În cadrul paginii `ClassView`, efectuați un dublu clic pe funcția constructor `CMDICoinDoc`.
6. Apăsăți **Ctrl+W** pentru a apela `ClassWizard` sau selectați **ClassWizard** din meniul **View**.
7. Introduceți următoarea linie de cod după comentariile `TODO` din funcția constructor:

```
m_nCoins = 1;
```

`CMDICoinDoc` conține acum o variabilă membru protejată, `m_nCoins`, care este inițializată în funcția constructor. Observați că inițializarea variabilelor în funcția constructor a documentului nu ridică probleme în cazul modelului MDI, ceea ce nu este valabil și pentru aplicațiile SDI. Într-o aplicație MDI, o nouă instanță de obiect document este creată de fiecare dată când se creează sau se deschide un document, fiind distrusă la închiderea documentului. Într-o aplicație SDI, obiectul document este creat o singură dată, fiind apoi refolosit pentru alte documente; obiectul este distrus la încheierea aplicației.

Ați adăugat, de asemenea, o funcție membru de acces, `GetCoinCount`, care va fi folosită pentru a furniza valoarea `m_nCoins`.

VEDEȚI...

- Pentru mai multe informații despre operațiile de inițializare în cadrul aplicațiilor SDI, vedeți pagina 260.

Protejarea variabilelor documentului

Este bine ca clasa document să conțină variabile membru protejate a căror accesare să se efectueze prin intermediul unor funcții membru publice.

Accesarea datelor documentului din cadrul reprezentării

Pentru ca reprezentarea să poată extrage date ale documentului, ea trebuie să fie mai întâi capabilă să acceseze obiectul document. Infrastructura MFC se ocupă automat de acest aspect, adăugând în clasa reprezentare a aplicației o funcție `GetDocument`. Această funcție întoarce un pointer către obiectul document la care reprezentarea a fost atașată prin intermediul machetei de document.

Clasa `CMDICoinView` afișează teancul de monede în cadrul funcției `OnDraw`. Infrastructura MFC apelează funcția `OnDraw` ori de câte ori reprezentarea trebuie afișată pe ecran. Editați funcția `CMDICoinView::OnDraw` așa cum o arată listingul 21.2.

LISTINGUL 21.2 LST22_1.CPP - Accesarea datelor documentului pentru generarea reprezentării

```
1 void CMDICoinView::OnDraw(CDC* pDC)
2 {
3     CMDICoinDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5
6     // TODO: add draw code for native data here
7     // ** Salvam pensula curenta
8     CBrush* pOldBrush = pDC->GetCurrentBrush();
9
10    // ** Cream o pensula plina de culoare galbena
11    CBrush br;
12    br.CreateSolidBrush(RGB(255,255,0));
13
14    // ** Selectam pensula galbena in contextul dispozitiv
15    pDC->SelectObject(&br);
16
17    // ** Obtinem de la document numarul de monede si desenam
18    // ** cate doua elipse care sa reprezinte fiecare moneda
19    for(int nCount = 0; nCount < pDoc->GetCoinCount();
20        nCount++)
21    {
22        int y = 100 - 10 * nCount;
23        pDC->Ellipse(40, y, 100, y-30);
24        pDC->Ellipse(40, y+10, 100, y-35);
25    }
26    // ** Selectam la loc pensula originala
27    pDC->SelectObject(pOldBrush);
28 }
```

Primele două linii ale funcției (liniile 3 și 4) sunt generate automat de AppWizard. Funcția `GetDocument` întoarce un pointer la obiectul document asociat reprezentării. Pointerul este

folosit în bucla `for` din linia 19 pentru a obține numărul curent de monede, fiind apelată funcția `GetCoinCount`.

VEDEȚI...

- Pentru informații privind contextele dispozitiv, vedeți pagina 323.
- Pentru informații privind generarea de grafică, vedeți pagina 372.

Modificarea datelor documentului și actualizarea reprezentării

Pentru a putea modifica numărul de monede, trebuie să adăugați în exemplul `MDICoin` două opțiuni de meniu – una care să adauge o monedă și cealaltă care să înlăture o monedă. La selectarea lor, aceste opțiuni vor apela fiecare câte o funcție a clasei document care va crește sau va scădea numărul de monede și va asigura actualizarea tuturor reprezentărilor asociate documentului. Opțiunile de meniu trebuie adăugate în cadrul resursei `IDR_MDICOITYPE`, deoarece aceasta este specifică tipului de document. Pentru adăugarea opțiunilor de meniu urmați pașii de mai jos.

Adăugarea opțiunilor de meniu

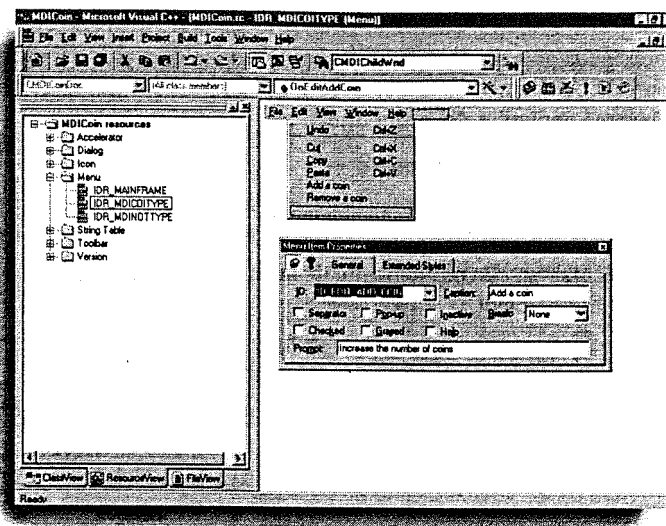
1. În cadrul paginii `ResourceView`, expandați catalogul `Menu` și efectuați un dublu clic pe `IDR_MDICOITYPE`. Resursa meniu va fi afișată în editorul de resurse.
2. Efectuați un clic pe elementul de meniu **E**dit afișat în editorul de resurse. Este deschis meniul derulant.
3. Efectuați un dublu clic pe elementul gol de la baza meniului derulant. Caseta de dialog `Menu Item Properties` este afișată ca în figura 21.7.
4. Specificați un identificator (**ID**) pentru elementul de meniu. Pentru cazul de față introduceți `ID_EDIT_ADD_COIN`.
5. Specificați o etichetă (**C**aption) pentru elementul de meniu. Pentru cazul de față introduceți `Add a Coin`.
6. Specificați o explicație (**P**rompt) pentru elementul de meniu. Pentru cazul de față introduceți `Increase the number of coins`.
7. Efectuați un dublu clic pe elementul gol de la baza meniului derulant.
8. Specificați un identificator pentru elementul de meniu. Pentru cazul de față introduceți `ID_EDIT_REMOVE_COIN`.
9. Specificați o etichetă pentru elementul de meniu. Pentru cazul de față introduceți `Remove a Coin`.
10. Specificați o explicație pentru elementul de meniu. Pentru cazul de față introduceți `Decrease the number of coins`.
11. Apăsăți **Ctrl+W** pentru a apela `ClassWizard` sau selectați **C**lass**W**izard din meniul **V**iew.
12. Selectați pagina `Message Maps`.
13. Selectați `CMDICoinDoc` în caseta combinată `Class Name`.
14. Selectați `ID_EDIT_ADD_COIN` în caseta cu listă `Object IDs`.

15. Selectați **COMMAND** în caseta cu listă **Messages**, după care efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
16. Selectați **ID_EDIT_REMOVE_COIN** în caseta cu listă **Object IDs**.
17. Selectați **COMMAND** în caseta cu listă **Messages**, după care efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
18. Efectuați un clic pe butonul **Edit Code**.

Modificarea datelor documentului trebuie reflectată în cadrul reprezentărilor asociate. În acest scop sunt apelate funcții care determină redesenarea reprezentărilor. Editați cele două noi funcții de tratare, `OnEditAddCoin` și `OnEditRemoveCoin`, așa cum o arată listingul 21.3.

FIGURA 21.7

Caseta de dialog **Menu Item Properties**.



LISTINGUL 21.3 LST22_2.CPP – Actualizarea reprezentărilor asociate pe baza documentului

```

1 void CMDICoinDoc::OnEditAddCoin()
2 {
3     // TODO: Add your command code handler here
4
5     // ** Incrementam numarul de monede
6     m_nCoins++;
7
8     // ** Actualizam reprezentarea pentru a redesena teancul de
9     // monede
10    UpdateAllViews(NULL);
11 }
12 void CMDICoinDoc::OnEditRemoveCoin()

```

```

13 {
14     // TODO: Add your command code handler here
15
16     // ** Decrementam numarul de monede
17     if(m_nCoins > 0)
18         m_nCoins--;
19
20     // ** Actualizam reprezentarea pentru a redesena teancul de
21     // monede
22     UpdateAllViews(NULL);
23 }

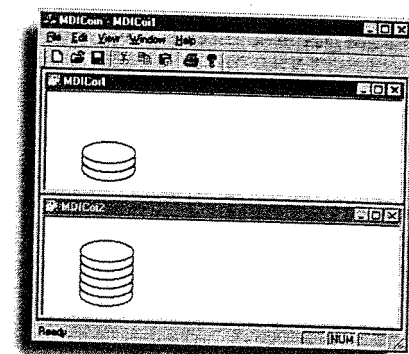
```

Precum vedeți, cele două funcții incrementează și, respectiv, decrementează variabila `m_nCoins` (liniile 6 și 18). Apoi este apelată funcția `UpdateAllViews`, fiind transmis parametrul `NULL` (liniile 9 și 21). Această valoare `NULL` determină actualizarea tuturor reprezentărilor atașate la document. Actualizarea fiecărei reprezentări se face prin apelul funcției `OnUpdate` a acesteia. Efectul este invalidarea zonei client a ferestrei reprezentării și apelarea funcției `OnDraw`.

Acum asamblați și rulați proiectul **MDICoin**. Încercați să selectați opțiunea **New** a meniului **File** pentru a crea o nouă fereastră document, după care experimentați selectarea opțiunilor de meniu **Add a Coin** și **Remove a Coin**. Reprezentarea afișată ar trebui să fie similară cu cea din figura 21.8. Veți putea, de asemenea, să creați o a doua reprezentare a aceluiași document, selectând opțiunea **New Window** a meniului **Window**.

FIGURA 21.8

Aplicația **MDICoin**.



Adăugarea de noi machete de document

Avantajul pe care modelul MDI îl aduce față de SDI nu privește doar posibilitatea oferită utilizatorului de a lucra simultan cu mai multe documente, ci și capacitatea de a gestiona într-o singură aplicație mai multe tipuri de documente. În acest sens va trebui să creați propriile clase document și reprezentare, să inserați resursele necesare și să adăugați codul care creează și inițializează un obiect `CMultiDocTemplate`. În secțiunea de față veți

adăuga la proiectul MDICoin un nou tip de document, care va opera nu cu monede, ci cu bancnote. Pentru generarea noilor clase document și reprezentare urmați pașii de mai jos.

Crearea noilor clase document și reprezentare

1. Apăsați Ctrl+W pentru a apela ClassWizard, sau selectați **ClassWizard** din meniul **View**.
2. Efectuați un clic pe butonul **Add Class** și selectați **New** din lista afișată. Este deschisă caseta de dialog **New Class**.
3. Introduceți în câmpul **Name** numele noii clase. Pentru exemplul nostru, precizați numele **CMDIBillDoc**.
4. Selectați **CDocument** din caseta combinată **Base Class**. Apoi efectuați un clic pe **OK** pentru a crea clasa.
5. Efectuați un clic pe butonul **Add Class** și selectați **New** din lista afișată. Este deschisă casetă de dialog **New Class**.
6. Introduceți în câmpul **Name** numele noii clase. Pentru exemplul nostru, precizați numele **CMDIBillView**.
7. Selectați **CView** din caseta combinată **Base Class**. Apoi efectuați un clic pe **OK** pentru a crea clasa.
8. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ClassWizard**.
9. Selectați clasa **CMDIBillDoc** în cadrul paginii **ClassView**; apoi selectați opțiunea **Add Member Variable** a meniului de context. Va fi afișată caseta de dialog **Add Member Variable**.
10. Introduceți **int** în caseta de editare **Variable Type**. Introduceți **m_nBills** în caseta de editare **Variable Name** și selectați butonul de opțiune **Protected Access**. Efectuați clic pe **OK**.
11. Având clasa **CMDIBillDoc** încă selectată, alegeți opțiunea **Add Member Function** a meniului de context.
12. Introduceți **int** în caseta de editare **Function Type**. Introduceți **GetBillCount** în caseta de editare **Function Declaration** și selectați butonul de opțiune **Public Access**. Efectuați clic pe **OK**.
13. Introduceți în corpul funcției **GetBillCount** linia de cod următoare:

```
return m_nBills;
```
14. În cadrul paginii **ClassView**, efectuați un dublu clic pe funcția constructor **CMDIBillDoc**.
15. Introduceți următoarea linie de cod după comentariile **TODO** din funcția constructor:

```
m_nBills = 1;
```
16. Selectați clasa **CMDIBillView** în cadrul paginii **ClassView**; apoi selectați opțiunea **Add Member Variable** a meniului de context.
17. Introduceți **CMDIBillDoc*** în caseta de editare **Function Type**. Introduceți **GetDocument** în caseta de editare **Function Declaration** și selectați butonul de opțiune **Public Access**. Efectuați clic pe **OK**.

18. Introduceți următoarea linie de cod în corpul funcției **GetDocument**:
- ```
return (CMDIBillDoc*)m_pDocument;
```

Acum sunt create noile clase document și reprezentare. Documentul conține un membru care va reține datele, precum și o metodă de acces. Clasa reprezentare conține o metodă care furnizează documentul asociat. Rețineți că va trebui să adăugați la începutul fișierului **MDIBillView.cpp**, înainte de directiva **#include** pentru **MDIBillView.h**, directiva **#include** următoare:

```
#include "MDIBillDoc.h"
```

Pașul următor este să adăugați o resursă de tip meniu. Pentru a crea această resursă prin copierea uneia existente, urmați pașii de mai jos:

1. În cadrul paginii **ResourceView**, expandați catalogul **Menu** și selectați elementul **IDR\_MDICOITYPE**. Apăsați Ctrl+C și apoi apăsați Ctrl+V. Va fi adăugată o copie a meniului selectat, identificatorul acesteia fiind **IDR\_MDICOITYPE1**.
2. Selectați **IDR\_MDICOITYPE1** în cadrul paginii **ResourceView**; apoi selectați **Properties** din meniul de context. Va fi afișată caseta de dialog **Menu Properties**.
3. Introduceți un identificator pentru meniu. Pentru exemplul nostru, fie acesta **IDR\_MDIBILTYPE**. Închideți caseta de dialog **Menu Properties**.
4. Efectuați un dublu clic pe **IDR\_MDIBILTYPE** în cadrul paginii **ResourceView**. Meniul va fi deschis în editorul de resurse.
5. Efectuați un clic pe elementul de meniu **Edit** din cadrul editorului de resurse. Va fi afișat meniul derulant corespunzător.
6. Efectuați un dublu clic pe elementul **Add a Coin**. Va fi afișată caseta de dialog **Menu Item Properties**.
7. Specificați un identificator (**ID**) pentru elementul de meniu. În cazul de față, introduceți **ID\_EDIT\_ADD\_BILL**.
8. Specificați o etichetă (**Caption**) pentru elementul de meniu. În cazul de față, introduceți **Add a Bill**.
9. Specificați un mesaj explicativ (**Prompt**) pentru elementul de meniu. În cazul de față, introduceți **Increase the number of bills**. Apoi efectuați un dublu clic pe elementul **Remove a Coin**.
10. Specificați un identificator pentru elementul de meniu. În cazul de față, introduceți **ID\_EDIT\_REMOVE\_BILL**.
11. Specificați o etichetă pentru elementul de meniu. În cazul de față, introduceți **Remove a Bill**.
12. Specificați un mesaj explicativ pentru elementul de meniu. În cazul de față, introduceți **Decrease the number of bills**.
13. Selectați **Save** din meniul **File**. Apoi selectați **Close** din meniul **File**.
14. Apăsați Ctrl+W pentru a apela ClassWizard, sau selectați **ClassWizard** din meniul **View**, iar apoi accesați pagina **Message Maps**.

15. Selectați `CMDIBillDoc` în caseta combinată **Class Name**.
16. Selectați `ID_EDIT_ADD_BILL` în caseta cu listă **Object IDs**.
17. Selectați `COMMAND` în caseta cu listă **Messages**; apoi efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
18. Selectați `ID_EDIT_REMOVE_BILL` în caseta cu listă **Object IDs**.
19. Selectați `COMMAND` în caseta cu listă **Messages**; apoi efectuați un clic pe butonul **Add Function**. Efectuați un clic pe **OK** în caseta de dialog **Add Member Function**.
20. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ClassWizard**.

Acum, clasa `CMDIBillDoc` conține două noi funcții de tratare a comenzilor de meniu, `OnEditAddBill` și `OnEditRemoveBill`. Aceste două funcții trebuie să incrementeze și, respectiv, să decrementeze `m_nBills`. Folosind ca exemplu codul prezentat în listingul 21.3, editați cele două noi funcții (înlocuiți `m_nCoins` cu `m_nBills`). După ce ați modificat ambele funcții, pasul următor este să creați o resursă șir de caractere corespunzătoare documentului.

#### Crearea unui șir de caractere corespunzător documentului

1. În cadrul paginii **ResourceView**, expandați catalogul **String Table** și apoi efectuați un dublu clic pe **String Table**. Tabela de șiruri va fi deschisă în editorul de resurse.
2. Pentru a insera noul șir sub șirul `IDR_MDICOITYPE`, selectați `IDR_MDICOITYPE` din tabela de șiruri și alegeți opțiunea **New String** a meniului de context. Va fi afișată caseta de dialog **String Properties**.
3. Specificați un identificator (**ID**) pentru noul șir. În cazul de față, introduceți `IDR_MDIBILTYPE`.
4. Specificați conținutul șirului (**Caption**). În cazul de față, introduceți șirul următor:  
`\nMDIBil\nBills\n\n\nMDIBill.Document\nMDIBil Document`
5. Editați șirul `IDR_MAINFRAME` pentru a deveni `MDICoins&Bills`.
6. Editați șirul `IDR_MDICOITYPE` pentru a deveni  
`\nMDIBil\nCoins\n\n\nMDIBil.Document\nMDIBil Document`
7. Închideți caseta de dialog **String Properties**.

Șirul corespunzător documentului are un format bine definit, identic pentru aplicațiile SDI și MDI. Șirul este format din șapte sub-șiruri, separate prin caracterul `\n` (linie nouă). Semnificația fiecărui sub-șir este descrisă în tabelul 21.3.

**TABELUL 21.3 Resursa șir de caractere asociată documentului**

| Șir | Exemplu | Descriere                                                                                                                                               |
|-----|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   |         | Șir vid în MDI. În cazul SDI, acesta reprezintă titlul aplicației. În cazul MDI, titlul aplicației este reținut separat în <code>IDR_MAINFRAME</code> . |

| Șir | Exemplu                          | Descriere                                                                                                                                                                                                                                                                               |
|-----|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2   | <code>MDIBil</code>              | Numele implicit al unui document. Pentru fiecare nou document se atașează automat un număr de identificare. De exemplu, în MS Word acest șir este <code>Document</code> , astfel că primul document nou este <code>Document1</code> , al doilea este <code>Document2</code> și tot așa. |
| 3   | <code>Bills</code>               | Titlul documentului. Folosit pentru a prezenta utilizatorului lista cu documentele disponibile. Puteți lăsa acest sub-șir vid dacă doriți să ascundeți utilizatorului existența documentului.                                                                                           |
| 4   | <code>Bill Viewer (*.bil)</code> | Descrierea tipului de document și a filtrului de fișiere. Utilizată în caseta de dialog standard <b>File Open</b> , în caseta combinată <b>Files of Type</b> .                                                                                                                          |
| 5   | <code>.bil</code>                | Extensia fișierelor care conțin astfel de documente. Rețineți că acest șir nu trebuie să conțină caractere wildcard.                                                                                                                                                                    |
| 6   | <code>MDIBil.Document</code>     | Numele de înregistrare în Explorer. Folosit pentru a crea o asociere pentru tipul de document în Explorer, astfel încât deschiderea unui fișier corespunzător să ducă la lansarea aplicației.                                                                                           |
| 7   | <code>MDIBil Document</code>     | Numele de identificare în registru și pentru OLE. Acest șir este înscris în registru pentru a activa suportul OLE. Utilizatorii văd acest șir atunci când selectează comanda <b>OLE Insert Object</b> .                                                                                 |

Totul este pregătit acum pentru utilizarea noilor clase document și reprezentare din cadrul proiectului `MDICoin`. Puteți să lăsați infrastructura MFC să se ocupe de instanțierea obiectelor și de gestionarea ferestrelor reprezentare și cadru. Nu mai trebuie decât să înregistrați în cadrul aplicației perechea document/reprezentare și cadrul asociat, creând în acest scop o a doua machetă de document. Machetele de document se înregistrează în supradefiniția funcției `CWinApp::InitInstance`. Pentru a crea și adăuga noua machetă de document, adăugați liniile de cod de la 16 la 25, prezentate în listingul 21.4. Rețineți că la începutul fișierului `MDICoinApp.cpp` va trebui să adăugați următoarele directive de includere:

```
#include "MDIBillDoc.h"
#include "MDIBillView.h"
```

**LISTINGUL 21.4 LST22\_3.CPP – Crearea și inițializarea machetei de document**

```
1 BOOL CMDICoinApp::InitInstance()
2 {
3 // ** Nota: unele linii au fost eliminate pentru concizie
4
5 // Register the application's document templates.
 Document templates
```

(continuare)

## LISTINGUL 21.4 Continuare

```

6 // serve as the connection between documents,
 frame windows, and views.
7
8 CMultiDocTemplate* pDocTemplate;
9 pDocTemplate = new CMultiDocTemplate(
10 IDR_MDICOITYPE,
11 RUNTIME_CLASS(CMDICoinDoc), — ❶
12 RUNTIME_CLASS(CChildFrame), // custom MDI child frame
13 RUNTIME_CLASS(CMDICoinView));
14 AddDocTemplate(pDocTemplate);
15
16 // ** Instantiem o a doua macheta de document si o
17 // ** initializam transmitand identificatorul pentru resursele
18 // ** asociate documentului si informatii dinamice pentru
 clasele document, cadru si reprezentare.
19 // ** Apoi apelam AddDocTemplate pentru a inregistra macheta
 in cadrul aplicatiei.
20 pDocTemplate = new CMultiDocTemplate(
21 IDR_MDIBILTYPE,
22 RUNTIME_CLASS(CMDIBillDoc),
23 RUNTIME_CLASS(CChildFrame), // custom MDI child frame — ❷
24 RUNTIME_CLASS(CMDIBillView));
25 AddDocTemplate(pDocTemplate);
26
27 // create main MDI Frame window
28 CMainFrame* pMainFrame = new CMainFrame;
29 if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
30 return FALSE;
31 m_pMainWnd = pMainFrame;
32
33 // ** Nota: unele linii au fost eliminate pentru concizie
34 }

```

❶ Macheta documentului original, creată de AppWizard.

❷ Adăugăm în aplicație un al doilea obiect machetă de document.

În fine, trebuie să implementați în clasa reprezentare codul care va afișa teancul de bancnote în funcție de numărul acestora reținut de către document. Editați funcția `CMDIBillView::OnDraw` în concordanță cu listingul 21.5.

## LISTINGUL 21.5 LST22\_4.CPP – Implementarea codului de desenare în cadrul clasei reprezentare

```

1 void CMDIBillView::OnDraw(CDC* pDC)
2 {
3 // ** Obținem un pointer la document
4 CMDICoinDoc* pDoc = GetDocument(); — ❶
5 ASSERT_VALID(pDoc);
6
7 // ** Salvăm pensula curentă
8 CBrush* pOldBrush = pDC->GetCurrentBrush();
9
10 // ** Cream o pensula plină de culoare verde
11 CBrush br;
12 br.CreateSolidBrush(RGB(0,128,32));
13
14 // ** Selectăm pensula verde în contextul dispozitiv
15 pDC->SelectObject(&br);
16
17 // ** Obținem de la document numărul de bancnote și desenăm
18 // ** câte un dreptunghi și un $ prin care reprezentăm fiecare
 bancnotă
19 for(int nCount = 0; nCount < pDoc->GetBillCount(); — ❷
 nCount++)
20 {
21 int x = 40 + 20 * nCount;
22 pDC->Rectangle(x, 40, x+100, 90);
23 pDC->TextOut(x + 5, 45, "$");
24 }
25
26 // ** Selectăm la loc pensula originală
27 pDC->SelectObject(pOldBrush);
28 }

```

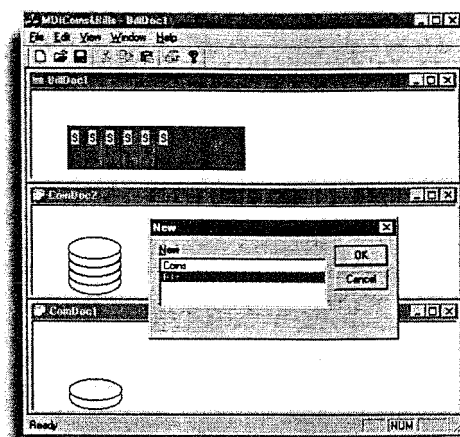
❶ Obținem un pointer la obiectul document.

❷ Apelăm funcția de acces a documentului care furnizează numărul curent de bancnote.

Funcția `GetDocument` întoarce un pointer la obiectul document, acesta fiind reținut în `pDoc`. Pointerul este folosit în bucla `for` din linia 19 pentru a obține numărul curent de monede prin apelul funcției `GetBillCount`. Codul de desenare produce o imitație ieftină de bancnote de dolari, folosind o pensulă verde pentru a umple uniform un dreptunghi și afișând simbolul \$.

Asamblați și rulați proiectul. La selectarea opțiunii **New** a meniului **File** ar trebui să fie afișată caseta de dialog **New**. Încercați să deschideți ambele tipuri de documente și apoi comutați între ele; observați că meniul **Edit** se modifică în funcție de documentul activ. Remarcați, de asemenea, că barele de titlu ale aplicației și documentului sunt generate pe baza resursei șir asociată tipului de document. Figura 21.9 înfățișează o ipostază a aplicației MDICoin.

FIGURA 21.9  
Aplicația MDICoin.



#### VEDEȚI...

- Pentru mai multe informații despre meniuri, vedeți pagina 270.
- Pentru mai multe informații despre contextele dispozitiv, vedeți pagina 323.
- Pentru a afla mai multe despre generarea de grafică, vedeți pagina 372.

## CAPITOLUL

# 22

## Tipărirea și previzualizarea

**Personalizarea infrastructurii standard în scopul unei implementări facile a tipăririi și a previzualizării**

**Implementarea rapoartelor pe mai multe pagini**

**Gestionarea diferitelor rapoarte de proporție și dimensiuni ale dispozitivelor**

**Tipărirea direct prin intermediul mecanismului de imprimare din Windows, cu scurt-circuitarea infrastructurii standard**

## Utilizarea funcționalității oferite de infrastructură

Infrastructurile SDI și MDI create de către AppWizard conțin în mod implicit un schelet de suport pentru tipărire și previzualizare. Puteți să renunțați la acest suport, deselectând opțiunea **Printing and Print Preview** din pasul 4 al casetei de dialog MFC AppWizard, dar în general se dovedește util în orice proiect și consumul suplimentar de resurse este minim. Cea mai mare parte a procesului efectiv de tipărire cade în sarcina contextului dispozitiv și a GDI (despre care am discutat în capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”). Infrastructura vă oferă un context dispozitiv pentru tipărirea unei pagini de document; în bună măsură, acesta poate fi tratat asemănător cu un context dispozitiv al unei ferestre obișnuite.

### Independența de dispozitiv

Până la apariția Windows, multe sisteme de operare nu dispuneau de drivere dispozitiv pentru imprimante care să permită producerea de grafică. Acest lucru a obligat producătorii de aplicații să scrie drivere dispozitiv specifice propriilor aplicații pentru imprimantele mai răspândite. Prin urmare, fiecare aplicație era însoțită de propria serie de drivere dispozitiv pentru aceleași modele de imprimante, rezultatul fiind o imensă cantitate de cod redundant. Din fericire, modelul de context dispozitiv propus de Windows ne permite să tratăm ecranul, imprimanta sau plotterul într-o aceeași manieră, ca pe o suprafață de desenare bidimensională. Mai mult decât atât, un driver dispozitiv nu mai este scris decât o singură dată, chiar de către producătorul dispozitivului.

### VEDEȚI...

- Pentru mai multe informații despre infrastructura SDI, vedeți pagina 246.
- Pentru explicații amănunțite privind contextele dispozitiv, vedeți pagina 323.

## Utilizarea facilităților de tipărire implicite

Infrastructura SDI (interfață de tip document singular) permite tipărirea imaginilor afișate în cadrul reprezentărilor pe baza informațiilor aflate în document. Cum aceste informații sunt deja afișate de către reprezentările aplicației, este probabil să poată fi tipărite prin adăugarea în cadrul reprezentărilor a unui suport adecvat.

Pentru afișarea unei imagini, infrastructura apelează funcția `OnDraw()` a reprezentării. Există o funcție similară, `OnPrint()`, care este apelată pentru a permite reprezentării să efectueze tipărirea informațiilor. De multe ori, această operație se reduce la a utiliza codul de desenare implementat în funcția `OnDraw()`. În acest caz, nu mai este nevoie să implementați funcția `OnPrint()`; infrastructura rezolvă implicit situația în clasa de bază `CView` și apelează `OnDraw()`. Imprimanta este tratată atunci asemeni ecranului, oferind funcțiilor de desenare un context dispozitiv care îl înlocuiește pe cel obișnuit, asociat ecranului. Funcția `OnDraw()` poate determina dacă a primit un context dispozitiv pentru ecran sau pentru imprimantă, dar cum funcțiile de desenare se comportă la fel în ambele cazuri, nici măcar această informație nu este necesară.

Puteți să cercetați funcționalitatea oferită de infrastructura standard creând o aplicație SDI tipică prin intermediul AppWizard. Lăsați selectată opțiunea **Printing and Print Preview** din pasul 4 (ceea ce înseamnă că puteți să efectuați clic pe butonul **Finish** din pasul 1) și denumiți proiectul `PrintIt`.

### Suportul standard pentru tipărire oferit de infrastructură

Suportul standard pentru tipărire și previzualizare este disponibil doar în aplicațiile SDI și MDI. Aplicațiile de tip dialog trebuie să implementeze propriul suport pentru tipărire.

Primul lucru de care aveți nevoie este o imagine pe care să o tipăriți. Puteți să creați o imagine de test în funcția `OnDraw()` a clasei `CPrintItView` (o clasă obișnuită, derivată din `CView`), așa cum se ilustrează în listingul 22.1. Aici se generează o imagine stilizată formată din linii, în centrul său aflându-se un text afișat cu un font mare (vedeți figura 22.1). Imaginea de test nu contează prea mult, dar va permite o comparație sugestivă între ce se generează pe hârtie și, respectiv, pe ecran.

### LISTINGUL 22.1 LST21\_1.CPP – Crearea unei imagini de test în cadrul funcției `OnDraw`

```
1 void CPrintItView::OnDraw(CDC* pDC)
2 {
3 CPrintItDoc* pDoc = GetDocument();
4 ASSERT_VALID(pDoc);
5
6 // TODO: add draw code for native data here
7
8 // ** Stabilim maparea metrica
9 pDC->SetMapMode(MM_LOMETRIC); ①
10
11 // ** Declaram si cream un font cu inaltimea de 2,2 cm
12 CFont fnBig;
13 fnBig.CreateFont(220,0,0,0,FW_HEAVY,FALSE,FALSE,0,
14 ANSI_CHARSET,OUT_DEFAULT_PRECIS,
15 CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
16 FF_SWISS+VARIABLE_PITCH,"Arial");
17
18 // ** Selectam noul font, retinandu-l pe cel original
19 CFont* pOldFont = pDC->SelectObject(&fnBig);
20
21 // ** Declaram un dreptunghi client
22 CRect rcClient;
23 GetClientRect(&rcClient);
24
25 // ** Il convertim in unitati logice
26 pDC->DPtoLP(&rcClient);
27
28 // ** Pregatim cateva variabile pentru desenare
29 const int nPoints = 50;
30 int xm = rcClient.Width();
```

(continuare)



```

31 int ym = rcClient.Height();
32 double dAspW = xm/(double)nPoints;
33 double dAspH = ym/(double)nPoints;
34
35 // ** Selectam o penita neagra
36 CPen* pOldPen =
37 (CPen*)pDC->SelectStockObject(BLACK_PEN);
38
39 // ** Trasam liniile
40 for(int i=0;i<nPoints;i++)
41 {
42 int x0 = (int)(i * dAspW);
43 int y0 = (int)(i * dAspH);
44
45 pDC->MoveTo(x0,0);
46 pDC->LineTo(xm,y0);
47 pDC->LineTo(xm-x0,ym);
48 pDC->LineTo(0,ym-y0);
49 pDC->LineTo(x0,0);
50 }
51
52 // ** Selectam la loc penita originala
53 pDC->SelectObject(pOldPen);
54
55 // ** Textul este afisat in raport cu o linie de baza
56 pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
57 pDC->SetBkMode(TRANSPARENT);
58
59 // ** Afisam textul gri
60 pDC->SetTextColor(RGB(64,64,64));
61 pDC->TextOut(xm/2,ym/2,"Sample Print");
62
63 // ** Selectam la loc fontul original
64 pDC->SelectObject(pOldFont);
65 }

```

❶ Alegem modul de mapare MM\_LOMETRIC, așa că unitatea de coordonate logică este o zecime de milimetru.

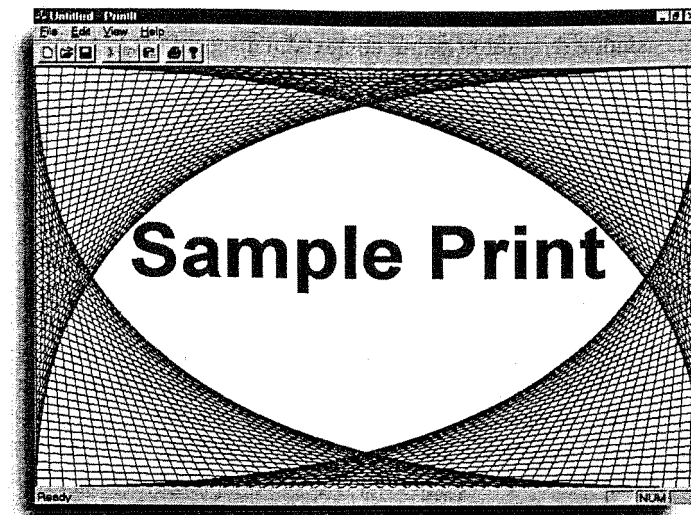
❷ Aceste linii formează rețeaua din fundal.

❸ Linia 61 are ca efect afișarea textului Sample Print în centrul ferestrei.

Deși funcția `OnDraw()` este relativ întinsă, codul său nu prezintă aspecte deosebite. Sunt trasate linii pe suprafața zonei client, iar în centru este afișat un text. Observați, în linia 9, că modul de mapare este `MM_LOMETRIC`; în consecință, unitatea logică de coordonate măsoară o zecime de milimetru (vedeți capitolul 15, „Despre desenarea în cadrul contextelor dispozitiv”).

FIGURA 22.1

Imagine de test afișată de  
Printit într-o fereastră.



În linia 13 este creat un font cu o înălțime de 2,2 cm, acesta fiind folosit la afișarea de text din linia 61. Liniile de la 40 la 50 desenează cadrul reticular, bazându-se pe coordonatele dreptunghiului client. Vă las pe dumneavoastră să descifrați detaliile; ceea ce ne interesează aici este să studiem tipărirea.

#### Utilizarea modurilor de mapare la afișarea pe ecran

Ați putea observa că, la afișarea pe ecran, înălțimea fontului nu măsoară exact 2,2 cm. Aceasta se întâmplă deoarece este dificil pentru Windows să cunoască exact dimensiunea fiecărui tip de monitor (chiar dacă este selectat corect în cadrul panoului de control). În schimb, driverul dispozitiv de imprimantă are o mai bună precizie a dimensiunilor, astfel că fontul tipărit ar trebui să aibă înălțimea de exact 2,2 cm.

Dacă asamblați și rulați programul după ce adăugați în funcția `OnDraw()` codul din listingul 22.1, imaginea afișată în fereastra aplicației va fi cea înfățișată în figura 22.1.

Așadar, întrebarea este: Ce trebuie să faceți ca să tipăriți această imagine? Surprinzător de puțin, pentru că infrastructura standard se ocupă de tipărire, apelând funcția `OnDraw()` și transmițându-i contextul dispozitiv al imprimantei, în locul celui asociat ferestrei.

Dacă efectuați un clic pe meniul **File** al aplicației **PrintIt** și selectați opțiunea **Print Preview**, veți vedea o mică reprezentare a imaginii plasată în colțul din stânga sus, în timp ce fontul apare prea mare pentru rețeaua de linii. Nu este vina infrastructurii; aceasta a făcut tot ce a putut pentru o cât mai bună reprezentare a ferestrei, dar coordonatele pe care le-a primit sunt inadecvate pentru contextul dispozitiv. Problema pornește de la apelul `GetClientRect()` din linia 23.

Observați că `GetClientRect()` este un membru al reprezentării, nu al contextului dispozitiv. Acest fapt nu ridică nici o problemă la afișarea în fereastră, deoarece contextul

dispozitiv are aceleași dimensiuni cu zona client a ferestrei. Aici, însă, folosiți dimensiunile ferestrei în cadrul contextului dispozitiv al imprimantei (acestea fiind mici prin comparație), în timp înălțimea de 2,2 cm cu care ați creat fontul nu este afectată (din cauza modului de mapare).

#### VEDEȚI...

- Pentru mai multe detalii privind supradefinirea funcției `OnDraw()`, vedeți pagina 338.
- Pentru mai multe informații despre trasarea de linii, vedeți pagina 357.
- Pentru mai multe informații despre afișarea de text, vedeți pagina 379.

## Supradefinirea funcției `OnPrint()`

Pentru a rezolva problema cu dimensiunile dreptunghiului client, va trebui să utilizați dreptunghiul adecvat pentru imprimantă, în locul celui corespunzător ferestrei. Din fericire, infrastructura apelează o funcție virtuală pe care puteți să o supradefiniți în cadrul reprezentării și să o utilizați pentru a obține toate informațiile necesare. Așa cum ați aflat mai devreme, numele acestei funcții este `OnPrint()` și ea este analogă cu `OnDraw()`. Pentru desenarea într-o fereastră este apelată `OnDraw()`; pentru desenarea la imprimantă este apelată `OnPrint()`. Poate că vă întrebați de ce a fost executat codul funcției `OnDraw()` la previzualizarea pentru tipărire a imaginii. Implementarea implicită din `CView` a funcției `OnPrint()` apelează pur și simplu `OnDraw()`, transmitând contextul dispozitiv pentru tipărire.

#### Funcțiile virtuale și polimorfismul

O funcție virtuală este o funcție care este definită într-o clasă de bază și a cărei implementare efectuează de obicei o acțiune implicită. O clasă derivată din această clasă de bază poate eventual să implementeze o versiune proprie a funcției virtuale, care să efectueze o operație mai specifică. La fiecare apel al funcției, chiar și prin intermediul unui pointer de tipul clasei de bază la obiectul derivat, va fi executată implementarea din clasă derivată; această tehnică se numește supradefinirea funcțiilor virtuale. Funcția supradefinită poate, totodată, să apeleze versiunea aflată în clasă de bază, specificând explicit în apel domeniul clasei de bază. În acest fel, funcția supradefinită poate să folosească funcționalitatea oferită de clasă de bază. Funcțiile virtuale reprezintă una din modalitățile principale prin care programatorii de C++ implementează polimorfismul obiectelor.

Dar nu este obligatoriu ca `OnPrint()` să apeleze `OnDraw()`; puteți supradefini funcția `OnPrint()` pentru a desena cu totul altceva, deși majoritatea aplicațiilor tipăresc ceea ce afișează pentru utilizator. Aceste aplicații re folosesc codul `OnDraw()` împreună cu contextul dispozitiv pentru tipărire.

#### Supradefinirea funcției virtuale `OnPrint()`

1. Efectuați un clic pe pagina `ClassView` din secțiunea spațiului de lucru al proiectului.
2. Efectuați un clic pe semnul plus din partea superioară pentru a afișa clasele proiectului.
3. Efectuați un clic cu butonul drept pe clasa reprezentare în care doriți să supradefiniți funcția `OnPrint()` (`CPrintItView`, în exemplul `PrintIt`) pentru a apela meniul de context.

4. Selectați opțiunea **Add Virtual Function** pentru a afișa caseta de dialog `New Virtual Override`.
5. În lista **New Virtual Functions** ar trebui să figureze funcția virtuală `OnPrint`.
6. Efectuați un clic pe butonul **Add and Edit** pentru a începe editarea funcției virtuale `OnPrint()`.

Implementarea standard pentru supradefiniția funcției `OnPrint()` arată astfel:

```
void CPrintItView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
 // TODO: Add your specialized code here
 CView::OnPrint(pDC, pInfo);
}
```

Primul lucru pe care-l puteți observa este că se deosebește de `OnDraw()` prin existența unui al doilea parametru, un pointer `pInfo` la un obiect de tip `CPrintInfo`. Aici veți găsi toate detaliile legate de tipărire, inclusiv coordonatele de care aveți nevoie pentru dreptunghiul corespunzător contextului dispozitiv de tipărire. Există multe variabile membru ale clasei `CPrintInfo` care sunt utile. Câteva dintre ele sunt prezentate în tabelul 22.1.

#### Utilizarea implementării implicite din clasa `CView` a funcției `OnPrint()`

Observați că implementarea generată pentru supradefiniția funcției `OnPrint()` apelează versiunea `OnPrint()` din clasă de bază, `CView`. Dacă inspecți codul sursă MFC veți descoperi minunea prin care infrastructura oferă suportul automat pentru tipărire. În fișierul sursă `..MFC\SRV\VIEWCORE.CPP`, aflat în directorul `Visual C++`, puteți găsi implementarea pentru `CView::OnPrint()`, care se rezumă la a apela `OnDraw()`, transmitând pointerul `pDC` la contextul dispozitiv.

**TABELUL 22.1 Variabile membru ale clasei `CPrintInfo` care conțin informații legate de tipărire**

| Numele variabilei               | Valoare                                                                                                             |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <code>m_nCurPage</code>         | Numărul paginii curente, pentru tipărirea pe mai multe pagini                                                       |
| <code>m_nNumPreviewPages</code> | 1 sau 2, în funcție de numărul de pagini afișate la previzualizare                                                  |
| <code>m_rectDraw</code>         | Coordonatele dreptunghiului care reprezintă pagina de tipărit                                                       |
| <code>m_pPD</code>              | Pointer la un obiect <code>CPrintDialog</code> , în cazul în care este folosită caseta de dialog <code>Print</code> |
| <code>m_bDirect</code>          | TRUE în cazul în care caseta de dialog <code>Print</code> nu a mai fost apelată                                     |
| <code>m_bPreview</code>         | TRUE pentru previzualizare                                                                                          |
| <code>m_strPageDesc</code>      | Un șir de format care ajută la generarea numerelor de pagină                                                        |
| <code>m_lpUserData</code>       | Un pointer care poate fi utilizat pentru a reține date ale utilizatorului                                           |

Despre alte variabile membru ale clasei `CPrintInfo` vom discuta mai târziu în acest capitol, dar mai întâi trebuie să determinați coordonatele dreptunghiului de tipărire (și nu ale dreptunghiului ferestrei). Membrul `m_rectDraw` conține coordonatele dreptunghiului

care reprezintă pagina curentă de tipărit. Acestea sunt coordonatele care trebuie folosite în funcția `OnDraw()` împreună cu contextul dispozitiv de tipărire. Există, totuși, o problemă, și anume faptul că această structură nu este transmisă funcției `OnDraw()`; puteți, însă, să copiați coordonatele într-o variabilă membru a clasei `CPrintItView`.

Pentru a reține coordonatele dreptunghiului, adăugați liniile următoare după comentariul `// TODO`, dar înainte de apelul `CView::OnPrint()`:

```
// ** copiem dreptunghiul de tiparire aflat in pInfo
if (pInfo) m_rcPrintRect = pInfo->m_rectDraw;
```

Astfel, dreptunghiul de tipărire va fi reținut în membrul `m_rcPrintRect` al clasei `CPrintItView`. În consecință, trebuie să declarați această variabilă, lucru ușor de făcut dacă efectuați un clic cu butonul drept pe clasa `CPrintItView` în cadrul paginii `ClassView` din secțiunea spațiului de lucru al proiectului și selectați opțiunea **Add Member Variable**. În câmpul **Variable Type** introduceți `CRect`, iar numele variabilei este, evident, `m_rcPrintRect`. Tipul de acces ar trebui să fie privat, pentru că nu aveți nevoie și nici nu doriți ca alte clase să acceseze acest dreptunghi intern.

#### Accesul public, privat și protejat

Pentru fiecare funcție membru și variabilă membru din definiția unei clase poate fi specificat un acces public, privat sau protejat. Acesta are efect asupra modului în care alte clase și clasele derivate pot accesa aceste funcții și variabile membru. Accesul public permite accesul oricărei clase la variabila sau funcția membru. Accesul privat permite numai funcțiilor membru ale aceleiași clase să acceseze membrul în cauză. Accesul protejat este similar cu accesul privat, cu deosebirea că clasele derivate din clasa de bază cu acces de tip public pot, de asemenea, să acceseze membrii protejați ai clasei de bază.

Aceste tehnici permit programatorilor de C++ să impună măsuri de siguranță, permițând accesarea obiectelor exclusiv prin intermediul unor metode de acces adecvate și „publice”. În acest fel scade probabilitatea accesării accidentale a unei variabile sau funcții membru care ar fi putut duce la blocarea aplicației.

#### VEDEȚI...

➤ Pentru mai multe amănunte despre clasele reprezentare și despre `CView`, vedeți pagina 260.

### Utilizarea contextului dispozitiv de tipărire

Contextul dispozitiv primit de `OnPrint()` diferă puțin de contextul dispozitiv de afișare prin aceea că ar putea avea mai puține culori și este probabil mai mare decât ecranul. Lăsând la o parte aceste atribute, îl puteți folosi la desenare asemeni contextului dispozitiv de afișare. În acest fel, aceeași funcție `OnDraw()` poate fi utilizată nu numai pentru afișarea în fereastră, ci și pentru tipărire. Exact așa este implementată funcția `CView::OnPrint()` din clasa de bază.

Contextul dispozitiv conține un indicator pe care-l puteți inspecta prin intermediul funcției `IsPrinting()` pentru a determina dacă urmează să desenați într-un context dispozitiv de afișare sau într-unul de tipărire. Puteți să țineți cont de această diferență pentru a genera grafică distinctă pentru imprimantă și pentru ecran sau, mai elegant, puteți să adaptați pentru tipărire coordonatele folosite.

În cazul exemplului nostru, este suficient ca în funcția `OnDraw()` să folosiți pentru tipărire coordonatele `m_rcPrintRect`. Codul care apelează funcția `IsPrinting()` pentru a determina dacă trebuie folosit dreptunghiul client al ferestrei sau dreptunghiul de tipărire este prezentat în listingul 22.2.

#### LISTINGUL 22.2 LST23\_2.CPP – Utilizarea dreptunghiului de tipărire în implementarea `OnDraw()`

```
1 // Declaram un dreptunghi client
2 CRect rcClient;
3
4 // ** Verificam daca avem un context dispozitiv de tiparire
5 if (pDC->IsPrinting())
6 {
7 // ** Tiparim, asa ca folosim dreptunghiul de tiparire
8 rcClient = m_rcPrintRect;
9 }
10 else
11 {
12 // ** Nu tiparim, deci avem nevoie de dreptunghiul client
13 GetClientRect(&rcClient);
14 }
15
16 // Il convertim in unitati logice
17 pDC->DPtoLP(&rcClient);
```

❶ Această linie transformă coordonatele dispozitiv, exprimate în pixeli, în coordonate logice, exprimate în zecimi de milimetru.

Observați că linia 5 a listingului 22.2 conține o instrucțiune `if` în care este apelată funcția `IsPrinting()` a contextului dispozitiv. Această funcție întoarce `TRUE` în cazul în care contextul dispozitiv este destinat tipăririi (sau previzualizării) și `FALSE` în cazul oricărui alt context dispozitiv. Dacă este un context dispozitiv de tipărire, puteți să înregistrați în `rcClient` dreptunghiul de încadrare a paginii tipărite pe care l-ați reținut anterior, așa cum se vede în linia 8. În cazul afișării în fereastră, veți folosi ca de obicei funcția `GetClientRect()` pentru a determina zona client, așa cum o arată linia 13.

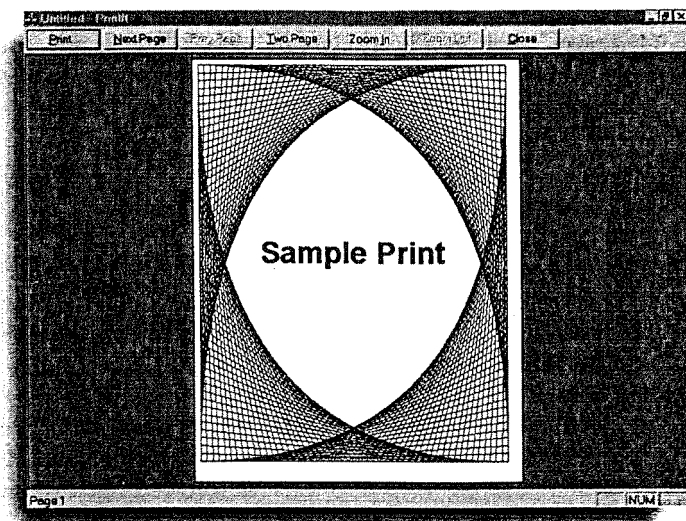
Pentru că ați folosit un mod de mapare, coordonatele dreptunghiului, fie acesta de tipărire sau de afișare, trebuie transformate din unități dispozitiv în unități logice. În acest scop este apelată funcția `DPtoLP()`, în linia 17. Dacă modificați funcția `OnDraw()` existentă și adăugați liniile 4-14, după asamblarea și rularea aplicației veți obține rezultate mult mai bune la previzualizare (vedeți figura 22.2).

#### VEDEȚI...

➤ Pentru a înțelege mai bine modurile de mapare, vedeți pagina 338.

FIGURA 22.2

Previzualizarea la tipărire atunci când sunt folosite coordonatele dreptunghiului care încadrează pagina de tipărire.



## Păstrarea proporției dimensiunilor

Așa cum puteți vedea în figura 22.2, din cauză că foaia de hârtie este mai lungă și mai puțin lată decât fereastra, imaginea generată este alungită. Raportul dintre lățime și înălțime se numește *proporția dimensiunilor*. Pentru a împiedica alungirea imaginii într-o direcție sau alta trebuie să păstrați aceeași proporție a dimensiunilor care caracterizează și imaginea din fereastră. Codul din listingul 22.2 nu încearcă să păstreze proporția dimensiunilor, ceea ce de cele mai multe ori nu poate să convină, așa că va trebui să modificați codul pentru a conserva această proporție în cazul imaginii tipărite.

Cea mai bună strategie pe care o puteți folosi în acest caz este să determinați care dimensiune dintre lățime și înălțime poate fi egală cu dimensiunea corespunzătoare a paginii, asigurând astfel o acoperire maximă, după care veți ajusta cealaltă dimensiune astfel încât să se păstreze proporția dimensiunilor.

### Proporția dimensiunilor în cadrul dispozitivelor

Pentru cele mai multe imprimante veți găsi probabil că proporția dimensiunilor este 1:1. Dar dacă priviți cu atenție o foaie tipărită cu o imprimantă termică, așa cum este cazul faxurilor, veți vedea clar că proporția dimensiunilor unui pixel este alta.

În acest scop aveți nevoie de informații privind dimensiunile paginii și proporția acestora. Aceste amănunte (și multe altele) sunt furnizate de o funcție a contextului dispozitiv numită `GetDeviceCaps()`. Dacă transmiteți acestei funcții indicatorii `ASPECTX` și `ASPECTY` veți putea determina raportul dintre lățimea și înălțimea unui pixel. Dacă acest raport este 1:1, pixelul este pătrat; altfel, pixelul este alungit și ar putea diferi de proporția dimensiunilor pentru ecran. Dacă diferă, puteți decide care axă vă oferă imaginea cea mai

mare, păstrând totodată aceeași proporție a dimensiunilor. În acest fel veți evita producerea unei imagini alungite.

Codul din funcția `OnDraw()` care tratează acest aspect este prezentat în listingul 22.3.

### LISTINGUL 22.3 LST23\_3.CPP – Generarea la tipărire a celei mai mari reprezentări posibile cu păstrarea proporției dimensiunilor

```
1 // ** Declaram un dreptunghi client si il determinam
2 CRect rcClient;
3 GetClientRect(&rcClient);
4
5 // ** Verificam daca avem un context dispozitiv de tiparire
6 if (pDC->IsPrinting())
7 {
8 // ** Calculam raportul dintre latimea de tiparire si cea a
9 // fereestrei
10 double dWidthRatio=(double)m_rcPrintRect.Width()/
11 (double)rcClient.Width();
12
13 // ** Calculam raportul dintre inaltimea de tiparire si cea a
14 // fereestrei
15 double dHeightRatio=(double)m_rcPrintRect.Height()/
16 (double)rcClient.Height();
17
18 // ** Calculam proportia dimensiunilor pentru dispozitiv
19 double dAspect=(double)pDC->GetDeviceCaps(ASPECTX)/
20 (double)pDC->GetDeviceCaps(ASPECTY);
21
22 // ** Determinam noua inaltime relativa
23 int nHeight=(int)(rcClient.Height() *
24 dHeightRatio * dAspect);
25
26 // ** Determinam noua latime relativa
27 int nWidth=(int)(rcClient.Width() *
28 dWidthRatio * dAspect);
29
30 // ** Copiem intregul dreptunghi
31 rcClient=m_rcPrintRect;
32
33 // ** Determinam cea mai buna potrivire (pe latime sau pe
34 // inaltime)
35 if (nHeight > nWidth)
36 {
37 // ** Pe inaltime este cel mai bine, asa ca ajustam
38 // latimea
```

(continuare)

## LISTINGUL 22.3 Continuare

```

35 rcClient.BottomRight().x=
36 m_rcPrintRect.TopLeft().x + nWidth;
37 }
38 else
39 {
40 // ** Pe latime este cel mai bine, asa ca ajustam
41 inaltimea
42 rcClient.BottomRight().y=
43 m_rcPrintRect.TopLeft().y + nHeight;
44 }
45
46 // Transformam in unitati logice
47 pDC->DPTOLP(&rcClient);

```

- ❶ Calculăm rapoartele dintre dimensiunile ferestrei și cele respective ale paginii de tipărire.
- ❷ Aici calculăm proporția dimensiunilor din cadrul imprimantei pe baza informațiilor obținute din caracteristicile contextului dispozitiv.
- ❸ Determinând dacă imaginea tipărită este mai lungă sau mai lată, puteți să utilizați optim pagina astfel încât să tipăriți cea mai mare reprezentare posibilă.

Observați că afișarea în fereastră și tipărirea folosesc ambele coordonatele ferestrei care au fost determinate prin apelul `GetClientRect()` din linia 3. În cazul afișării în fereastră, nu se mai întâmplă nimic altceva și codul continuă în mod obișnuit.

În schimb, se întâmplă multe în cazul tipăririi, când testul `IsPrinting()` din linia 6 întoarce valoarea `TRUE`. Mai întâi trebuie să determinați rapoartele dintre lățimea ferestrei și lățimea paginii și dintre înălțimea ferestrei și înălțimea paginii. Acestea sunt determinate în liniile 9 și 13, împărțind dimensiunea paginii la dimensiunea respectivă a ferestrei.

Următorul lucru care trebuie calculat este proporția dimensiunilor din cadrul dispozitivului. Funcția `GetDeviceCaps()` este apelată în linia 17 pentru a determina raportul dintre lățime și înălțime în cadrul dispozitivului, rezultatul fiind reținut în `dAspect`.

Având aceste elemente, puteți acum să calculați valorile comparative pentru lățime și înălțime în cadrul dispozitivului, în raport cu dimensiunile ferestrei, așa cum o arată liniile 21 și 25. Aceste calcule, care țin cont în mod corespunzător și de proporția dimensiunilor din cadrul dispozitivului, vor avea ca rezultate înălțimea ajustată în funcție de o lățime egală cu cea a paginii și viceversa. În continuare trebuie să decideți care dintre lățime și înălțime va lua valoarea maximă, ajustând apoi cealaltă dimensiune. Condiția din linia 32 ia această decizie comparând lățimea și înălțimea. Aceasta înseamnă că dacă aveți o fereastră înaltă și subțire, este mai bine să folosiți întreaga înălțime a paginii și să ajustați lățimea; reciproc, dacă aveți o fereastră scundă și lată, este mai bine să folosiți întreaga lățime a paginii și să ajustați înălțimea. În funcție de alegere, ajustarea este efectuată în linia 35 sau 42, înscrind în coordonata x sau y a colțului dreapta-jos lățimea sau, respectiv, înălțimea modificată.

## Opțiuni de tipărire uzuale oferite de aplicații

Multe aplicații de grafică oferă mai multe opțiuni prin care puteți să păstrați sau să ignorați proporția dimensiunilor. Exemplul nostru implementează o opțiune numită uzual `Best Fit` (cea mai bună potrivire). O altă opțiune tipică este `Size to Page`, care face ca imaginea să fie întinsă pe întreaga suprafață a paginii tipărite (modificând, în consecință, proporția dimensiunilor). S-ar putea să întâlniți și alte opțiuni, precum `Original Image Size`, care păstrează proporția dimensiunilor, dar afișează imaginea (de obicei centrată) la dimensiunea de pe ecran (de regulă, arată relativ mică pe o imprimantă de rezoluție înaltă).

Observați că toate celelalte dimensiuni din `rcClient` sunt cele ale hârtiei, fiind preluate prin atribuirea din linia 29, astfel că singura modificare necesară este ajustarea menționată. După toate acestea, programul continuă și va folosi pentru desenare dreptunghiul ajustat.

Dacă asamblați și rulați aplicația după adăugarea în funcția `OnDraw()` a liniilor din listingul 22.3, veți vedea că la tipărirea sau la previzualizarea ferestrei va fi păstrată proporția dimensiunilor acesteia. Dacă alungiți fereastra astfel încât înălțimea sa să depășească lățimea, imaginea generată pentru tipărire se va întinde pe întreaga înălțime a paginii, fără a mai acoperi lățimea, dar va păstra proporția corectă a dimensiunilor.

## VEDEȚI...

➤ Pentru mai multe amănunte privind caracteristicile dispozitivelor, vedeți pagina 342.

## Paginarea și orientarea

Tipărirea unei singure pagini care să reprezinte ceea ce se afișează într-o fereastră este o cerință uzuală, dar procesul de tipărire vizează în bună măsură generarea unor documente mari și sofisticate, întinse pe mai multe pagini, pe baza datelor complexe ale utilizatorului. Infrastructura vine din nou în ajutor și simplifică acest proces oferind o casetă de dialog standard pentru configurarea tipăririi (`Print Setup`) și un sistem de enumerare a paginilor care permite tipărirea și previzualizarea unei secvențe de pagini specificate.

## Tipărirea pe mai multe pagini a unei imagini

Uneori veți dori să tipăriți o imagine întinsă pe mai multe pagini. În acest scop va trebui să generați în cadrul contextului dispozitiv câte o porțiune de imagine pentru fiecare pagină. Clasa `CScrollView` vă poate ajuta aici, deoarece oferă `ScrollToPosition()` și permite scalări arbitrare. În orice caz, rămâne necesară pregătirea și poziționarea contextului dispozitiv de tipărire de la o pagină la alta.

## Stabilirea paginilor de început și sfârșit

Primele elemente de luat în considerație în cazul unui document pe mai multe pagini sunt pagina de început și cea de sfârșit, acestea indicând totodată numărul de pagini care vor fi tipărite. La începerea procesului de tipărire este apelată o funcție virtuală a clasei reprezentare. Această funcție este `OnPreparePrinting()` și ea primește un parametru, un obiect `pInfo` de tip `CPrintInfo`. Aici operați prima dată cu `CPrintInfo` și aici puteți efectua primele modificări corespunzător cu cerințele de tipărire. Funcția `OnPreparePrinting()` este generată automat de `AppWizard` la crearea aplicației, așa că nu

mai este nevoie să o adăugați. Pentru a vedea implementarea implicită, efectuați un clic pe membrul `OnPreparePrinting()` al clasei `CPrintItView` din cadrul paginii `ClassView`.

Iată ceea ce ar trebui să vedeți:

```

BOOL CPrintItView::OnPreparePrinting(CPrintInfo* pInfo)
{
 // default preparation
 return DoPreparePrinting(pInfo);
}

```

În mod implicit este apelată funcția `DoPreparePrinting()`, transmițându-se pointerul `pInfo` la obiectul `CPrintInfo` asociat tipăririi. `DoPreparePrinting()` pregătește contextul dispozitiv necesar și, în cazul în care operația este de tipărire (și nu de previzualizare), apelează caseta de dialog standard `Print`. Despre această casetă de dialog vom discuta mai pe larg în secțiunea următoare, dar deocamdată puteți să stabiliți secvența de pagini pentru tipărire, modificând obiectul `CPrintInfo` înainte de apelul `DoPreparePrinting()`.

În acest sens, adăugați liniile de mai jos după comentariul `// default preparation`.

```

pInfo->SetMinPage(2);
pInfo->SetMaxPage(8);

```

Aceste două funcții membru ale clasei `CPrintInfo` modifică obiectul indicat de `pInfo`, fixând începutul la pagina 2 cu `SetMinPage()` și sfârșitul la pagina 8 cu `SetMaxPage()`.

Acum, la tipărirea documentului, funcția `OnPrint()` va fi apelată de șapte ori. Singura diferență dintre aceste apeluri va fi conținutul variabilei membru `pInfo->m_nCurPage`, care va reține numărul paginii curente și va varia de la 2 la 8.

În funcției de tipul aplicației dezvoltate, procedeul folosit pentru a determina numărul de pagini va diferi. Dacă vindeți compact-discuri audio și vreți să tipăriți o broșură cu produsele oferite, probabil că veți tipări pe fiecare pagină coperta unui CD și o prezentare sumară a acestuia, ceea ce înseamnă că pentru 120 de CD-uri aveți nevoie de 120 de pagini. Dar dacă tipăriți certificate guvernamentale care conțin elemente de identificare și formatare, probabil că va trebui să măsurați înălțimea tuturor componentelor și să calculați numărul de pagini după o paginare prealabilă. În orice caz, odată determinat numărul de pagini, `OnPreparePrinting()` este locul în care veți înscrie această valoare în cadrul obiectului `CPrintInfo`.

#### Evitarea la tipărire a casetelor de dialog `Print`

Nu trebuie neapărat să deranjați utilizatorul cu caseta de dialog `Print`; apelarea acesteia poate fi evitată dacă atribuiți variabilei `pInfo->m_bDirect` valoarea `TRUE` în cadrul funcției `OnPreparePrinting()`.

Pentru a evidenția diferența dintre un document complet și tipărirea unei ferestre, puteți să implementați în funcția `OnPrint()` generarea unei imagini complet diferite de cea produsă în `OnDraw()`, așa cum o ilustrează listingul 22.4. În această implementare `OnPrint()` nu

mai este apelată funcția `CView::OnPrint()` din clasa de bază, ceea ce înseamnă că nu se mai efectuează apelul `OnDraw()` implicit. Așadar, implementarea care urmează face ca imaginea tipărită și cea afișată să fie complet diferite.

#### LISTINGUL 22.4 LST23\_4.CPP – Implementarea unei tipări pe mai multe pagini în `OnPrint()`

```

1 void CPrintItView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
2 {
3 // TODO: Add your specialized code here
4
5 // ** Cream si selectam un font
6 CFont fnTimes;
7 fnTimes.CreatePointFont(720, "Times New Roman", pDC); ①
8 CFont* pOldFont = (CFont*)pDC->SelectObject(&fnTimes);
9
10 // ** Cream si selectam o pensula
11 CBrush brHatch(HS_CROSS, RGB(64, 64, 64));
12 CBrush* pOldBrush =
13 (CBrush*)pDC->SelectObject(&brHatch);
14
15 // ** Construim textul de tiparit
16 CString strDocText;
17 strDocText.Format("Page Number %d",
18 pInfo->m_nCurPage); ②
19
20 pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
21
22 // ** Cream o serie de obiecte de tip punct utile
23 CPoint ptCenter=pInfo->m_rectDraw.CenterPoint();
24 CPoint ptTopLeft=pInfo->m_rectDraw.TopLeft();
25 CPoint ptBotRight=pInfo->m_rectDraw.BottomRight();
26
27 // ** Definim varfurile rombului
28 CPoint ptPolyArray[4]=
29 {
30 CPoint(ptTopLeft.x,ptCenter.y);
31 CPoint(ptCenter.x,ptTopLeft.y);
32 CPoint(ptBotRight.x,ptCenter.y);
33 CPoint(ptCenter.x,ptBotRight.y);
34 };
35
36 // ** Desenam romb
37 pDC->Polygon(ptPolyArray, 4);
38
39 // ** Generam textul

```

(continuare)



## LISTINGUL 22.4 Continuare

```

40 pDC->TextOut(ptCenter.x,ptCenter.y,strDocText); ④
41
42 // ** Selectam fontul si pensula originale
43 pDC->SelectObject(pOldFont);
44 pDC->SelectObject(pOldBrush);
45 }

```

- ① Creăm un font de 72 de puncte apelând `CreatePointFont()`.
- ② Determinăm numărul paginii curente din cadrul obiectului `CPrintInfo`.
- ③ Definim vectorul cu vârfurile rombului din fundal.
- ④ Producem textul care conține numărul paginii curente.

În liniile 6-12 ale listingului 22.4 sunt create resursele necesare pentru tipărire (un font și o pensulă). Rețineți că există un loc mai potrivit în acest sens, așa cum veți vedea mai târziu în acest capitol, în secțiunea „Crearea obiectelor GDI în `OnBeginPrinting()`”.

Numărul paginii curente este folosit pentru a schimba conținutul textual din fiecare pagină corespunzător cu poziția acesteia în cadrul documentului, așa cum se vede în linia 17. Într-o aplicație adevărată, probabil că veți folosi acest număr de pagină ca referință în cadrul documentului, pentru a găsi anumite date. În exemplul cu compact-discurile pe care l-am menționat mai devreme, numărul de pagină ar fi folosit pentru identificarea unui CD, iar funcțiile de desenare ar utiliza informațiile corespunzătoare. Nu avem la dispoziție spațiul necesar pentru a ilustra ceva atât de sofisticat, așa că am folosit numărul paginii curente din `pInfo->m_nCurPage` într-o manieră pur demonstrativă.

Liniile 22-37 se ocupă de generarea ca fundal a unui romb, iar linia 40 produce în centrul paginii textul care conține numărul paginii curente. Liniile 43-44 selectează la loc fontul și pensula originale.

## Implementarea MFC a previzualizării

Infrastructura MFC implementează previzualizarea prin intermediul unei clase aparte, nedocumentată, numită `CPreviewDC`. Acest context dispozitiv special se comportă din punctul de vedere al aplicației ca un context dispozitiv de tipărire, dar afișează imaginea pe ecran ca un context dispozitiv de afișare obișnuit. Acest mod de operare este determinat de necesitatea de a simula elemente specifice tipăririi, așa cum este paginarea. Deși clasa nu este documentată, puteți să studiați codul său sursă, aflat în fișierul `MFC\Src\DCPREV.CPP` din directorul Visual C++.

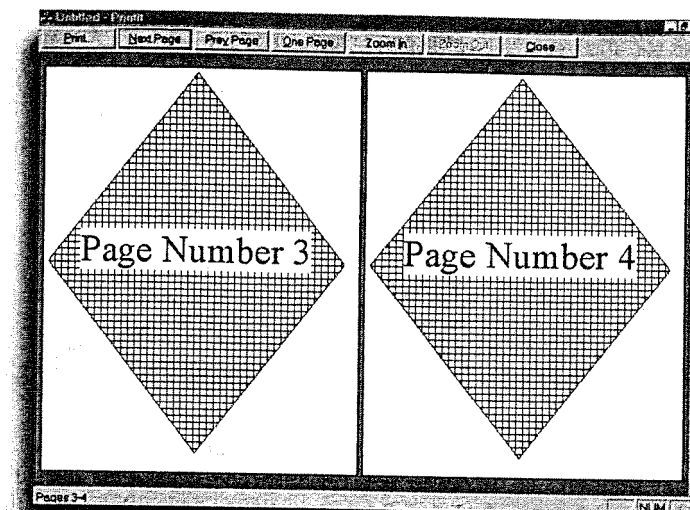
Dacă asamblați și rulați programul după efectuarea acestor modificări în cadrul funcției `OnPrint()` și apoi efectuați un clic pe meniul **File** și selectați **Print Preview**, veți putea să previzualizați mai multe pagini cu ajutorul butoanelor **Next Page** și **Prev Page**, înfățișate în figura 22.3. În cazul în care dispuneți de o imprimantă, veți putea să tipăriți documentul întins pe mai multe pagini.

## VEDEȚI...

- În legătură cu selectarea obiectelor în cadrul contextelor dispozitiv, vedeți pagina 323.
- În legătură cu desenarea cu ajutorul diferitelor tipuri de pensule, vedeți pagina 365.

FIGURA 22.3

Previzualizarea unui document pe mai multe pagini.



## Utilizarea casetei de dialog Print

Observați că la tipărirea unui document care conține mai multe pagini este afișată o casetă de dialog ce permite configurarea opțiunilor de tipărire, așa cum se vede în figura 22.4. Aceasta este caseta de dialog standard `Print`; ea este apelată din interiorul funcției `CView::DoPreparePrinting()`, apelată la rândul său din supradefiniția `OnPreparePrinting()`. Această casetă de dialog vă permite să stabiliți paginile de tipărit, numărul de exemplare, ordinea de aranjare, imprimanta destinație și o serie întreagă de elemente specifice imprimantei respective.

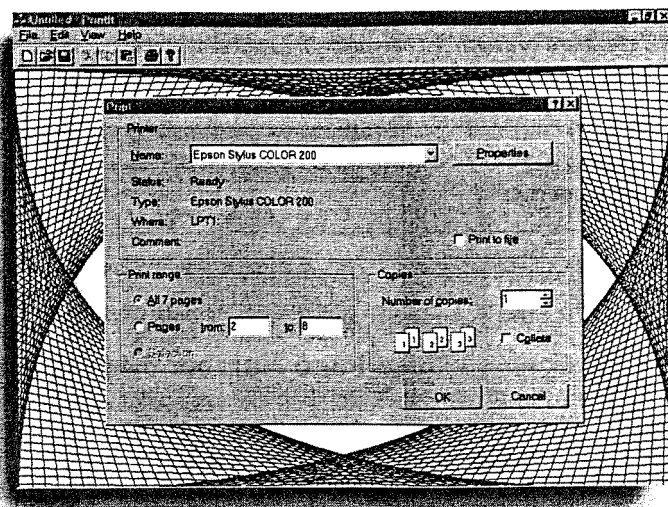
## Casetă de validare Collate

În cazul în care utilizatorul deselectează caseta de validare **Collate** din caseta de dialog `Print`, driverul de imprimantă va determina tipărirea la un loc a exemplarelor pentru fiecare pagină. Nu este nevoie de nici un cod aparte în acest sens – dar facilitatea trebuie suportată de driverul de imprimantă; altfel, ea va fi dezactivată și inaccesibilă în cadrul casetei de dialog `Print`.

Utilizatorul poate să modifice opțiunile de tipărire în cadrul acestei casete de dialog, ceea ce va duce la actualizarea informațiilor înscrise în obiectul `CPrintInfo` înainte de transmiterea sa către aplicație. Casetă de dialog poate fi personalizată mai mult sau mai puțin, în funcție de necesități și de efortul pe care sunteți dispus să-l investiți.



FIGURA 22.4  
Caseta de dialog standard  
Print.



Printre membrii clasei `CPrintInfo` prezentați în tabelul 22.1 se afla un pointer `m_pPD`. Pointerul indică un obiect de tip `CPrintDialog`, această clasă fiind o clasă MFC care încapsulează caseta de dialog Print. Aceeași clasă conține un membru `m_pd`, care este o structură `PRINTDLG` conținând opțiunile implicite afișate în caseta de dialog Print. Această structură conține mulți membri, prezentați în listingul 22.5. Se permite astfel o personalizare completă a casetei de dialog, mergând până la specificarea unei alte machete de dialog decât cea implicită (în caz că vă plac provocările). Nu avem spațiu suficient pentru a descrie în amănunt fiecare membru; una dintre variabilele membru cu semnificație mai evidentă este `nCopies`. Puteți să modificați numărul implicit de exemplare afișat în caseta de dialog, intervenind direct asupra membrului `nCopies` al acestei structuri înainte de apelul funcției `CView::DoPreparePrinting()`. În acest scop, adăugați linia următoare în cadrul funcției `OnPreparePrinting()`:

```
pInfo->m_pPD->m_pd.nCopies = 15;
```

#### Utilizarea structurii `DevMode`

Structura `DevMode` conține mai multe atribute utile care descriu caracteristicile tehnice și configurația dispozitivului. Un pointer la această structură este furnizat de funcția `GetDevMode()` a clasei `CPrintDialog`.

Odată adăugată această linie, la deschiderea casetei de dialog Print veți vedea că numărul de exemplare este implicit 15 (dacă imprimanta sau driverul de imprimantă suportă exemplările multiple). Puteți să modificați corespunzător și alte valori implicite din structura `PRINTDLG`.

#### LISTINGUL 22.5 LST23\_5.CPP – Structura `PRINTDLG`

```
1 typedef struct tagPD {
2 DWORD lStructSize;
3 HWND hwndOwner;
4 HANDLE hDevMode;
5 HANDLE hDevNames;
6 HDC hdc;
7 DWORD Flags;
8 WORD nFromPage;
9 WORD nToPage;
10 WORD nMinPage;
11 WORD nMaxPage;
12 WORD nCopies;
13 HINSTANCE hInstance;
14 DWORD lCustData;
15 LPPRINTHOOKPROC lpfnPrintHook;
16 LPSETUPHOOKPROC lpfnSetupHook;
17 LPCTSTR lpPrintTemplateName;
18 LPCTSTR lpSetupTemplateName;
19 HANDLE hPrintTemplate;
20 HANDLE hSetupTemplate;
21 } PRINTDLG;
```

❶ Ultimele șase variabile din structură vă permit să personalizați caseta de dialog Print prin înlocuirea celei standard cu propria machetă de dialog. Este o facilitare foarte sofisticată, care vă oferă un control complet asupra interfeței cu utilizatorul.

Odată ce utilizatorul a confirmat opțiunile din caseta de dialog Print prin intermediul butonului **OK**, acestea pot fi determinate cu ajutorul funcțiilor de acces ale clasei `CPrintDialog`, funcții prezentate în tabelul 22.2. Așadar, dacă vreți să aflați înainte de tipărire numărul de exemplare specificat de către utilizator, puteți să obțineți această valoare după revenirea din apelul `CView::DoPreparePrinting()`, așa cum o ilustrează listingul 22.6.

Firește, pot fi inspectate oricare dintre valorile înscrise în structura `PRINTDLG` indicată de `pInfo->m_pPD->m_pd`.

#### TABELUL 22.2 Funcțiile de acces ale clasei `CPrintDialog`

| Numele funcției              | Descriere                                                        |
|------------------------------|------------------------------------------------------------------|
| <code>GetCopies()</code>     | Întoarce numărul de exemplare specificat de utilizator           |
| <code>GetFromPage()</code>   | Întoarce pagina de început care a fost specificată               |
| <code>GetToPage()</code>     | Întoarce pagina de sfârșit care a fost specificată               |
| <code>GetPortName()</code>   | Întoarce portul de imprimantă selectat, cum ar fi LPT1:          |
| <code>GetDriverName()</code> | Întoarce driverul de imprimantă selectat (imprimanta destinație) |

(continuare)

TABELUL 22.2 Continuare

| Numele funcției  | Descriere                                              |
|------------------|--------------------------------------------------------|
| GetPrinterDC()   | Întoarce un context dispozitiv pentru imprimantă       |
| PrintAll()       | Întoarce TRUE dacă au fost selectate toate paginile    |
| PrintCollate()   | Întoarce TRUE dacă s-a solicitat aranjarea paginilor   |
| PrintRange()     | Întoarce TRUE dacă s-a specificat o secvență de pagini |
| PrintSelection() | Întoarce TRUE dacă au fost selectate pagini arbitrare  |

#### LISTINGUL 22.6 LST23\_6.CPP – Validarea casetei de dialog standard Print din punctul de vedere al numărului de exemplare specificat

```

1 BOOL CPrintItView::OnPreparePrinting(CPrintInfo* pInfo)
2 {
3 pInfo->SetMinPage(1);
4 pInfo->SetMaxPage(10);
5
6 pInfo->m_pPD->m_pd.nCopies = 3;
7
8 do
9 {
10 // ** Verificam daca utilizatorul a renuntat la tiparire
11 if (DoPreparePrinting(pInfo) == FALSE) — ❶
12 return FALSE;
13
14 // ** Avertizam utilizatorul daca a specificat prea multe
15 // exemplare
16 if (pInfo->m_pPD->GetCopies() > 5)
17 AfxMessageBox("Please choose less than
18 5 copies");
19
20 // ** Repetam pana la specificarea unei
21 // valori adecvate
22 } while (pInfo->m_pPD->GetCopies() > 5); — ❷
23 return TRUE;
24 }
```

❶ Trebuie să permitem utilizatorului să părăsească bucla dacă efectuează un clic pe butonul Cancel, pentru că altfel nu are de ales decât să tipărească.

❷ Această condiție neobișnuită obligă utilizatorul să specifice cel mult cinci exemplare. Se ilustrează, astfel, posibilitatea de validare din punctul de vedere al opțiunilor exprimate în caseta de dialog Print.

În listingul 22.6, CView::DoPreparePrinting() întoarce FALSE în cazul în care utilizatorul a apăsat butonul Cancel (liniile 11 și 12). În caz contrar, linia 15 verifică numărul de exemplare și se afișează un avertisment dacă s-au specificat mai mult de cinci

exemplare (criteriul meu arbitrar). Bucla se încheie cu linia 19, repetându-se până când utilizatorul introduce un număr de exemplare adecvat sau apasă butonul Cancel.

### Utilizarea orientărilor de tip portret și peisaj

Dacă efectuați un clic pe meniul **File** al aplicației și selectați opțiunea **Print Setup**, veți putea să modificați orientarea implicită a imprimantei. În caseta de dialog puteți să alegeți **Portrait** sau **Landscape**. Nu este nevoie să modificați codul pentru a tipări în mod peisaj; dacă selectați această orientare și apoi optați pentru o previzualizare, veți vedea cum contextul dispozitiv este adaptat paginii culcate pe o parte. Atât timp cât aplicația ține cont de membrul `rectDraw` al obiectului `CPrintInfo`, ar trebui să se rezolve automat problema tipăririi în mod peisaj.

#### Diferite dimensiuni de pagină

Utilizatorul poate să specifice diferite dimensiuni de pagină, de la plicuri până la A3. Pentru tratarea acestor opțiuni nu este nevoie de prelucrări speciale, deoarece contextul dispozitiv transmis aplicației va fi adaptat automat de către Windows la dimensiunile specificate. Variabila membru `rectDraw` a obiectului `CPrintInfo` va conține coordonatele care exprimă dimensiunile contextului dispozitiv.

### Crearea obiectelor GDI în *OnBeginPrinting()*

Așa cum am menționat mai devreme, codul din listingul 22.4 funcționează bine, dar există o alternativă mai bună pentru alocarea resurselor necesare. În acest moment, pentru fiecare pagină tipărită este apelată `OnPrint()`, iar aceasta generează pagina și creează de la zero toate resursele folosite. Probabil că nu se va produce o încetinire semnificativă în cazul unor documente simple, dar pentru rapoarte complexe, de dimensiuni mari, ar fi de preferat să pregătiți o serie de resurse și alte elemente o singură dată, la început. Apoi puteți să tipăriți paginile specificate, eliberând resursele la sfârșit.

Funcția virtuală `OnBeginPrinting()` este locul ideal pentru astfel de inițializări, iar funcția sa pereche, `OnEndPrinting()`, este potrivită pentru eliberarea acestor resurse. `OnBeginPrinting()` este apelată după `OnPreparePrinting()` și este primul loc în care apare contextul dispozitiv de tipărire. Acest context dispozitiv este cel care va fi utilizat în procesul de tipărire, așa că în acest moment puteți să pregătiți toate obiectele GDI și coordonatele de pagină folosite. Implementarea oferită automat de `ClassWizard` nu este decât o funcție goală:

```

void CPrintItView::OnBeginPrinting(CDC* /*pDC*/,
 CPrintInfo* /*pInfo*/)
{
 // TODO: add extra initialization before printing
}
```

Priviți cu atenție la această definiție de funcție. Observați că parametrii sunt comentați, ceea ce va duce la compilare la apariția unor mesaje de avertizare în legătură cu parametrii nefolosiți. Înainte de a-i utiliza va trebui să aveți grijă să înlăturați marcasele de comentarii.

Acum puteți să plasați în această funcție codul pentru crearea obiectelor GDI, evitând repetarea acestei operații pentru fiecare pagină:

```
m_fnTimes.CreatePointFont(720, "Times New Roman", pDC);
m_brHatch.CreateHatchBrush(HS_CROSS, RGB(64, 64, 64));
```

Observați că obiectele `fnTimes` și `brHatch` au fost prefixate cu `m_`; este o convenție de denumire care arată că obiectele în cauză nu sunt locale (încapsulate într-o funcție), ci au ca domeniu întreaga clasă (sunt încapsulate în clasă). Pentru că obiectele GDI vor fi accesate în `OnPrint()`, este nevoie să le adăugați la definiția clasei. Obiectele font și pensulă pot fi inserate în definiția clasei astfel:

```
protected:
 CFont m_fnTimes;
 CBrush m_brHatch;
```

#### Tipărirea color și monocrom

Observați că valoarea `COLORREF` furnizată de macrodefiniția `RGB(64, 64, 64)` stabilește o culoare gri închis pentru pensulă. Majoritatea imprimantelor monocrome reprezintă destul de bine nuanțele de gri printr-un procedeu de combinare a punctelor. Puteți să specificați valori `COLORREF` pentru orice culoare pe 24 de biți; contextul dispozitiv va determina automat cea mai bună reprezentare a culorii pe care o poate oferi imprimanta. Imprimantele color s-ar putea să fie capabile să reprezinte culoarea specificată, în timp ce imprimantele monocrome vor transforma culoarea într-o nuanță de gri.

Adăugarea obiectelor se poate face fie efectuând un dublu clic pe clasa `CPrintItView` în cadrul paginii `ClassView` și editând direct, fie prin intermediul casetei de dialog `Add Member Variable`.

De asemenea, remarcați că pensula hașurată nu mai este creată prin intermediul constructorului, ci cu ajutorul funcției `CreateHatchBrush()`. Motivul este faptul că deși pensula va exista pe toată durata de viață a reprezentării, va trebui să apelați `DeleteObject()` în cadrul funcției `OnEndPrinting()` astfel încât resursele GDI să fie eliberate între două procese de tipărire. Adăugați în funcția `OnEndPrinting()` codul care șterge obiectele GDI font și pensulă, așa cum este ilustrat aici:

```
m_fnTimes.DeleteObject();
m_brHatch.DeleteObject();
```

Tot ce mai rămâne de făcut este să eliminați obiectele GDI locale din funcția `OnPrint()` și să înlocuiți aparițiile lor cu variabilele membru corespunzătoare. În acest scop veți șterge declarațiile variabilelor locale, `CFont fnTimes` și `CBrush brHatch`, precum și funcțiile de creare, selectând în loc fontul și pensula deja create:

```
CFont* pOldFont = (CFont*)pDC->SelectObject(&m_fnTimes);
CBrush* pOldBrush = (CBrush*)pDC->SelectObject(&m_brHatch);
```

Dacă asamblați și rulați aplicația după efectuarea acestor modificări, probabil că nu veți sesiza nici o diferență. Funcționalitatea a rămas aceeași, dar tipărirea și previzualizarea ar

trebui să fie ceva mai rapide. Dacă ați avea un document complex, de 100 de pagini, care să folosească multe resurse GDI, cu siguranță ați aprecia ca utilă această tehnică în scopul accelerării tipăririi.

#### Utilizarea coordonatelor primite în `OnBeginPrinting()`

Ați putea fi tentat, de asemenea, să vă ocupați de stocarea coordonatelor în cadrul funcției `OnBeginPrinting()`. Nu veți reuși, deoarece membrul `m_rectDraw` al obiectului `CPrintInfo` nu a fost inițializat până la acest punct, ceea ce înseamnă că veți folosi coordonate arbitrare.

#### VEDEȚI...

- Pentru a afla despre crearea pensulelor GDI, vedeți pagina 365.
- Pentru a afla despre crearea fonturilor GDI, vedeți pagina 386.
- Pentru a afla despre inserarea unei variabile membru prin intermediul paginii `ClassView`, vedeți pagina 428.

## Personalizarea operației de pregătire a contextului dispozitiv

Înainte de funcțiile `OnDraw()` și `OnPrint()` este apelată funcția virtuală `OnPrepareDC()`, iar aceasta poate fi supradefinită în clasa reprezentare cu scopul efectuării în cadrul contextului dispozitiv a unor eventuale intervenții necesare ambelor funcții `OnDraw()` și `OnPrint()`. Ați putea dori să selectați moduri de mapare sau să stabiliți opțiuni de desenare în cadrul contextului dispozitiv care să fie valabile atât pentru afișare, cât și pentru tipărire. Supradefiniția nu este creată de către `AppWizard`, dar poate fi adăugată ușor prin intermediul casetei de dialog `Add Virtual Function`. În exemplul nostru, un lucru comun funcțiilor `OnDraw()` și `OnPrint()` este apelul funcției `SetTextAlign()` a contextului dispozitiv. Ați putea să plasați acest apel în interiorul funcției `OnPrepareDC()`, ca aici:

```
void CPrintItView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
 pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
}
```

Pot exista situații, mai ales la pregătirea unor tipăriri conforme cu `WYSIWYG`, în care să fie de preferat stabilirea modurilor de mapare și a coordonatelor de fereastră în cadrul unei aceleiași funcții, înainte de apelarea funcției de desenare sau de tipărire. `OnPrepareDC()` este locul potrivit pentru codul de inițializare specific contextului dispozitiv.

#### VEDEȚI...

- Pentru a afla despre parametrii contextelor dispozitiv, vedeți pagina 323.
- Pentru a înțelege modul de aliniere a textului, vedeți pagina 379.

## Suspendarea procesului de tipărire

O altă utilizare pentru `OnPrepareDC()` este apelul funcțiilor de control al imprimantei sau al altor funcții care privesc tipărirea documentului. Dacă aveți un document lung, ați putea să oferiți utilizatorului posibilitatea de a suspenda procesul de tipărire și a înceta tipărirea.

Funcția `AbortDoc()` a contextului dispozitiv suspendă tipărirea documentului în cadrul contextului dispozitiv de tipărire. Puteți să încercați acest lucru adăugând în funcția `OnPrepareDC()` liniile de mai jos, efectul fiind suspendarea tipării după trei pagini:

```
if (pDC->IsPrinting())
 if (pInfo->m_nCurPage==3) pDC->AbortDoc();
```

## Tipărirea directă, fără mijlocirea infrastructurii

Până acum, în acest capitol, am discutat despre suportul pentru tipărire oferit de către infrastructurile SDI și MDI. Acest suport se îmbină elegant în arhitectura Document/Reprezentare, dar sunt situații în care nu doriți decât un acces rapid și facil la o imprimantă sau în care nu dispuneți de serviciile infrastructurii – într-o aplicație de tip dialog, de pildă.

Suportul oferit de infrastructură maschează suportul de nivel scăzut pentru tipărire, care reprezintă fundamentul pentru toate operațiile de tipărire. Secțiunea de față prezintă modul în care funcționează acest suport și ilustrează utilizarea sa într-o exemplu de aplicație de tip dialog.

## Apelarea directă a casetei de dialog Print

Ați văzut în secțiunea anterioară, „Utilizarea casetei de dialog Print”, modul în care clasa `CPrintDialog` încapsulează dialogul Print și structura aferentă `PRINTDLG` și cum este apelată caseta de dialog în cadrul funcției `CView::DoPreparePrinting()`.

Aceeași casetă de dialog și aceeași clasă pot fi utilizate explicit pentru selectarea imprimantei și a opțiunilor asociate, în aceeași manieră în care ați folosi o casetă de dialog modală obișnuită. Puteți să utilizați aceleași funcții de acces pe care le-ați folosit mai devreme, în cadrul funcției `OnPreparePrinting()`, pentru a fixa paginile selectate sau numărul de exemplare.

Listingul 22.7 ilustrează folosirea directă a casetei de dialog Print în scopul configurării imprimantei într-o aplicație de tip dialog, tipărindu-se apoi un mic document pe baza opțiunilor exprimate în caseta de dialog.

### Utilizarea modului special `CreateDC`

În loc să afișați caseta de dialog Print, puteți să profitați de funcționalitatea acesteia pentru a crea un context dispozitiv pentru tipărire, configurat cu opțiuni implicite. Apoi veți putea să tipăriți în cadrul acestui context dispozitiv în maniera obișnuită. Pentru utilizarea acestui procedeu există o funcție membru, `CreatePrinterDC()`, a clasei `CPrintDialog`. De asemenea, puteți să schimbați opțiunile implicite, modificând valorile înscrise în structurile `DEVMODE` și `DEVNAMES` asociate. Funcția `GetDevMode()` furnizează un pointer la prima dintre aceste structuri, iar `DEVNAMES` este accesibilă prin intermediul variabilei membru `m_pd` a structurii `PRINTDLG`.

Apelați la `AppWizard` pentru a crea o aplicație de tip dialog numită `DlgPrint` și creați cu `ClassWizard` o funcție de tratare `OnOK()` care va implementa codul pentru tipărire, prezentat în listingul 22.7.

### LISTINGUL 22.7 LST23\_7.CPP – Tipărirea directă a unui document în funcția `OnOK` a unei aplicații de tip dialog

```
1 void CdlgPrintDlg::OnOK()
2 {
3 // TODO: Add extra validation here
4
5 // ** Construim un obiect CPrintDialog
6 CPrintDialog dlgPrint(FALSE, PD_ALLPAGES, this);
7
8 if (dlgPrint.DoModal() == IDOK)
9 {
10 // ** Atasam contextul dispozitiv de tipărire creat de
11 // ** dialog la un obiect CDC
12 CDC dcPrint;
13 dcPrint.Attach(dlgPrint.GetPrinterDC());
14
15 // ** Cream și initializăm o structura DOCINFO
16 DOCINFO myPrintJob;
17 myPrintJob.cbSize = sizeof(myPrintJob);
18 myPrintJob.lpszDocName = "MyPrintJob";
19 myPrintJob.lpszOutput = NULL;
20 myPrintJob.lpszDatatype = NULL;
21 myPrintJob.fwType = NULL;
22
23 // ** Începem tipărirea documentului
24 if (dcPrint.StartDoc(&myPrintJob) >= 0)
25 {
26 // ** Începem o pagină
27 dcPrint.StartPage();
28
29 // ** Desenăm
30 dcPrint.TextOut(0, 0, "My Small Print Job");
31
32 // ** Trimitem pagina
33 dcPrint.EndPage();
34
35 // ** Închidem documentul
36 dcPrint.EndDoc();
37 }
38
39 // ** Stergem contextul dispozitiv de tipărire
40 dcPrint.DeleteDC();
41 }
42
```

①

②

③

(continuare)

## LISTINGUL 22.7 Continuare

```

43 // ** Continuum cu implementarea standard OnOK
44 CDialog::OnOK();
45 }

```

- ❶ Inițializăm structura DOCINFO cu un minim de informații necesare pentru declanșarea unui proces de tipărire.
- ❷ Singura linie tipărită, încadrată de apelurile StartPage() și EndPage().
- ❸ Contextul dispozitiv creat de caseta de dialog Print trebuie șters.

Listingul 22.7 conține în linia 6 declarația unui obiect `dlgPrint` de tip `CPrintDialog`, constructorului clasei fiindu-i transmiși trei parametri. Primul parametru este un indicator de condiție care poate fi `TRUE`, caz în care se afișează caseta de dialog `Print Setup`, sau `FALSE`, caz în care este afișată caseta de dialog `Print`. Al doilea parametru este o combinație de indicatori prin care se pot configura parametrii casetei de dialog (sunt prea mulți indicatori pentru a-i mai aduce în discuție). Cel de-al treilea parametru este un pointer la fereastra părinte; în cazul de față, pointerul `this` din C++ precizează că fereastra părinte este caseta de dialog a aplicației.

Apelul `dlgPrint.DoModal()` din linia 8 afișează caseta de dialog. Dacă utilizatorul efectuează clic pe **OK**, începe tipărirea; altfel, totul se încheie.

Dacă utilizatorul a efectuat clic pe butonul **OK** al casetei de dialog `Print`, este creat un context dispozitiv de tipărire și, în linia 13, este atașat la un obiect `CDC` în scopul unei operări mai facile. Trebuie să aveți grijă să ștergeți contextul dispozitiv propriu-zis, așa cum o arată linia 40.

Adăugați funcția de tratare și implementarea sa prezentată în listing, asamblați și rulați aplicația și efectuați un clic pe butonul **OK** din caseta de dialog a aplicației pentru a executa noul cod.

### Utilizarea funcțiilor *StartDoc()* și *EndDoc()*

Contextul dispozitiv `CDC` conține multe funcții legate de tipărire. Pentru a iniția o operație de tipărire, `Windows` trebuie să creeze un document de acumulare în care să fie depus procesul de tipărire, trimițându-l către imprimantă atunci când acesta este complet. Funcția `StartDoc()` determină `Windows` să înceapă acumularea, iar funcția `EndDoc()` anunță sistemul că documentul este complet și poate fi trimis către imprimantă. Funcția `AbortDoc()`, pe care ați întâlnit-o mai devreme, întrerupe tipărirea și nu mai trimite procesul de tipărire către imprimantă, ci îl suspendă.

Listingul 22.7 apelează în linia 24 membrul `StartDoc()` al obiectului `dcPrint` de tip context dispozitiv de tipărire, transmițând un pointer la o structură `DOCINFO`. Această structură conține detalii despre un proces de tipărire. Singura informație pe care trebuie să o specificați este numele documentului de acumulare, stabilit în linia 18. Observați existența unui membru mai puțin obișnuit, `cbSize`, care reține dimensiunea structurii. Acestuia îi este atribuită valoarea furnizată de `sizeof(myPrintJob)` în linia 17. Așa ceva

se întâlnește frecvent la nivelul `Win32 API`, iar `DOCINFO` este o structură în vechiul stil C; `cbSize` este necesar pentru că există mai multe forme diferite ale structurii `DOCINFO` și singurul lucru prin care pot fi deosebite acestea este dimensiunea.

Odată apelată, `StartDoc()` va încerca să declanșeze procesul de tipărire, întorcând o valoare pozitivă în caz de succes. Există multe motive pentru care funcția ar putea să eșueze, precum lipsa spațiului de pe disc sau din memorie sau un driver de imprimantă defect, așa că este bine să verificați codul întors înainte de a continua tipărirea.

#### Urmărirea procesului de tipărire în Windows

Puteți să urmăriți evoluția unui proces de tipărire definind un punct de întrerupere în funcția `OnPrint()` sau după un apel `StartDoc()` și deschizând fereastra de stare a imprimantei din cadrul grupului **Printers**, aflat în submeniul **Settings** al meniului **Start** din `Windows`.

După tipărirea documentului trebuie să apelați `EndDoc()`, ca în linia 36, pentru a determina transmiterea sa către imprimantă.

#### VEDEȚI...

➤ Pentru a afla mai multe despre `API` și `Win32 API`, vedeți pagina 655.

### Utilizarea funcțiilor *StartPage()* și *EndPage()*

O altă pereche de funcții ale contextului dispozitiv de tipărire este formată din `StartPage()` și `EndPage()`. Funcția `StartPage()` este utilizată pentru inițializarea contextului dispozitiv în scopul tipăririi unei noi pagini. Aceasta va reseta o serie de elemente ale contextului dispozitiv, cum ar fi poziția curentă a cursorului grafic, și va marca începutul unei noi pagini în cadrul documentului de acumulare.

De regulă veți apela `StartPage()`, veți genera în cadrul contextului dispozitiv reprezentarea unei pagini și apoi veți apela `EndPage()` pentru a înscrie pagina în fișierul de acumulare, marcând sfârșitul ei.

În listingul 22.7, `StartPage()` este apelată în linia 27, urmată de un apel `TextOut()` solitar, care afișează un text pe pagină, și de apelul `EndPage()` din linia 33.

#### Trimiterea de coduri Escape specifice dispozitivului

Teoretic, nu ar trebui niciodată să știți ceva despre imprimanta conectată, dar există o modalitate prin care puteți să treceți peste toate mecanismele de context dispozitiv din `Windows` și să trimiteți secvențe de caractere direct către imprimantă.

Funcția `Escape()` a clasei `CDC` vă permite exact acest lucru, oferindu-vă posibilitatea de a trimite coduri Escape specifice imprimantei direct către aceasta. Codurile Escape sunt identificate printr-un cod de funcție și sunt fie specifice dispozitivului (transmise direct către dispozitiv), fie specifice `Windows` (traduse de către driverul de dispozitiv din `Windows`).

O astfel de abordare ar putea fi necesară în cazul în care o anumită facilități a dispozitivului nu este suportată de către mecanismele standard din `Windows`.

La apelul funcției `EndPage()`, în documentul de acumulare sunt înscrise codurile speciale pentru `Form Feed` (pagină nouă), marcându-se încheierea paginii curente. Această secvență de apeluri `StartPage()` și `EndPage()` se repetă pentru fiecare pagină a documentului, iar în final se apelează `EndDoc()` pentru a încheia procesul de tipărire. Contextul dispozitiv de tipărire poate fi folosit pentru desenare la fel cum a fost utilizată `OnPrint()` în aplicația SDI, între apelurile `StartPage()` și `EndPage()`. Aceleași funcții sunt apelate și de către infrastructura SDI, dar aceasta le maschează, rămânând vizibil apelul `OnPrint()` între apelurile care încep și încheie o pagină.

## CAPITOLUL

# 23

## Salvarea, încărcarea și transferul datelor

---

Salvarea și încărcarea datelor documentului

---

Crearea, citirea și scrierea fișierelor

---

Transferul datelor înspre și dinspre Clipboard



## Utilizarea serializării

Serializarea reprezintă o tehnică ce poate fi folosită pentru transformarea datelor aplicației într-o listă secvențială de elemente de date individuale, urmată de stocarea lor pe disc sau transferul către un alt program. Atunci când un program încarcă sau acceptă un transfer de date serializate, se citește fluxul de elemente de date și se reconstituie în memorie obiecte structurate și care pot interacționa.

Infrastructurile SDI și MDI oferă suport pentru serializare și manipularea fișierelor – tot ce aveți de făcut este să implementați serializarea datelor atunci când infrastructura o necesită.

### Crearea unei infrastructuri SDI care lucrează cu fișiere

Procedul obișnuit de creare a unui nou proiect prin intermediul AppWizard poate fi folosit și pentru generarea unei aplicații SDI care să creeze, să întrețină și să serializeze obiectele de date stocate într-o clasă derivată din `CDocument`. Singurele aspecte relevante pentru serializare pe care trebuie să le luați în considerație atunci când creați noul proiect sunt șirurile asociate machetei de document, privind tipul implicit de document.

## Manipularea fișierelor în aplicațiile de tip dialog

Aplicațiile de tip dialog nu au nevoie în mod normal să folosească serializarea – manipularea directă a fișierelor prin intermediul clasei `CFile` ar trebui să fie suficientă (vom discuta mai târziu în acest capitol). Cu toate acestea, aplicațiile de tip dialog pot folosi serializarea cu ajutorul clasei `CArchive`. În acest scop trebuie să implementați în cadrul casetei de dialog o interfață cu utilizatorul corespunzătoare, care să inițieze procesul de serializare.

Aceste șiruri asociate machetei de document pot fi stabilite în pasul 4 din MFC AppWizard, efectuând un clic pe butonul **Advanced**. Casetă de dialog Advanced Options care este afișată (vedeți figura 23.1) vă permite să stabiliți următoarele șiruri specifice documentului:

- **File Extension** – Puteți specifica extensia asociată fișierelor document, introducând în caseta de editare **File Extension** orice extensie alfanumerică doriți. Fișierele create la salvarea documentului și cele încărcate de către infrastructură vor avea atașată această extensie.
- **File Type ID** – Vă permite să specificați un tip de document care va fi înscris în registrul sistemului, astfel încât alte aplicații vor putea să asocieze fișierele de acest tip cu aplicația dumneavoastră. În mod normal veți accepta valoarea implicită, care se bazează pe numele aplicației. În consecință, aplicația va putea fi lansată automat pentru a încărca un fișier ca răspuns la efectuarea de către utilizator a unui dublu clic pe fișierul respectiv, pe suprafața de lucru sau în Windows Explorer.
- **Filter Name** – Puteți specifica un filtru de fișiere corespunzător extensiei stabilite.
- **File Type Name** – Asociat cu **File Type ID** prin aceea că este numele echivalent care va fi afișat atunci când alte aplicații descriu documentele de acest tip. Aceste nume sunt cele care apar atunci când efectuați un clic cu butonul drept pe suprafața de lucru și selectați opțiunea **New** a meniului contextual. Este afișată o listă cu tipurile de

fișiere înregistrate. Despre operația de înregistrare vom discuta mai târziu în acest capitol, în secțiunea „Înregistrarea tipurilor de documente”.

### Opțiunile privind tipul de document

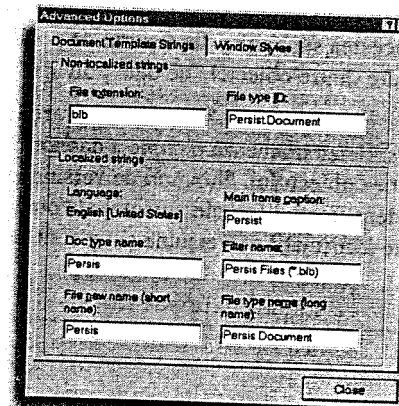
Opțiunile prezente în caseta de dialog Advanced Options sunt stocate sub forma resursei de tip șir `IDR_MAINFRAME`. Puteți, asadar, să modificați ulterior valorile stabilite, editând resursa șir `IDR_MAINFRAME` din tabela de șiruri prezentă în pagina **ResourceView** a secțiunii spațiului de lucru al proiectului.

- **File New Name** – Puteți specifica numele implicit care va fi utilizat la salvarea fișierelor fără nume.

Puteți să apelați AppWizard pentru a crea un exemplu de proiect SDI, numit **Persist**, care să illustreze serializarea. Dacă creați acest proiect, acceptați opțiunile propuse de MFC AppWizard în pașii de la 1 la 3, efectuând clic pe butonul **Next** pentru a avansa până la pasul 4. Aici efectuați un clic pe butonul **Advanced** pentru a afișa caseta de dialog Advanced Options și stabiliți extensia `blb` pentru numele de fișiere, așa cum se vede în figura 23.1.

FIGURA 23.1

Stabilirea extensiei de fișier în caseta de dialog Advanced Options.



### VEDEȚI...

- Pentru mai multe informații despre crearea aplicațiilor SDI, vedeți pagina 246.
- În legătură cu editarea resurselor de tip șir, vedeți pagina 45.

## Crearea obiectelor de date serializabile

Orice clasă derivată din clasa de bază `MFC CObject` are o funcție virtuală `Serialize()` pe care o puteți supradefini pentru a încărca și stoca variabilele membru ale clasei pe disc. Atunci când utilizatorul selectează **Save** sau **Open** din meniul **File** al unei aplicații SDI sau MDI, infrastructura va apela automat funcția `Serialize()` a documentului pentru a salva sau, respectiv, încărca obiectele de date ale documentului.

Clasa MFC `CArchive` este responsabilă pentru serializarea obiectelor derivate din `CObject` prin intermediul unei variabile membru de tip fișier a acestei clase.



Infrastructura construiește un obiect `CArchive` și-l atașează fișierului pe care utilizatorul îl încarcă sau îl salvează. Dumneavoastră puteți să supradefiniți funcția `Serialize()` în cadrul documentului (acesta fiind derivat din `CObject`). La salvare sau la încărcare, funcția `Serialize()` supradefinită va primi o referință, numită `ar`, care corespunde obiectului `CArchive` creat de către infrastructură. Acest obiect `CArchive` conține o variabilă de tip fișier care este legată la fișierul fizic de pe disc care este salvat sau încărcat. În cadrul funcției `Serialize()` a unui document trebuie să scrieți codul care apelează funcția `Serialize()` pentru fiecare din obiectele de date ale aplicației, transmitând fiecăruia obiectul `CArchive` (`ar`) pe care l-ați primit ca parametru de la infrastructură.

#### Transferul datelor în rețea prin intermediul `CArchive`

Clasa `CArchive` poate fi utilizată împreună cu clasele `CSocket` și `CSocketFile`. Această combinație vă permite să transferați date între două calculatoare aflate într-o rețea locală sau chiar într-o rețea de mare întindere, cum este Internetul.

Atunci când se salvează datele documentului, informațiile conținute în fiecare obiect sunt serializate și stocate în fișierul asociat cu obiectul arhivă. Atunci când se încarcă datele unui document, acestea sunt citite din fișierul de pe disc care este asociat cu obiectul arhivă și înscrise în fiecare dintre obiectele de date prin apelul propriilor funcții `Serialize()`.

Fișierul de pe disc creat în urma unui proces de serializare se numește fișier arhivă serializat, iar acest fișier stochează o copie a datelor fiecărui obiect din aplicație, împreună cu tipul și versiunea obiectului. Numărul de versiune vă permite să modificați datele obiectului și să identificați diferitele versiuni ale obiectului stocat pe măsură ce aplicația evoluează. Aceste informații de versiune și tip sunt cunoscute sub numele de *schemă*.

#### Schemele de serializare și schemele de baze de date

Deși terminologia este aceeași, schemele de serializare sunt cu totul altceva decât schemele de baze de date. O schemă de bază de date reține detalii pentru fiecare câmp din cadrul tabelor unei baze de date. În schimb, schemele de serializare rețin tipul clasei și un număr de versiune.

Puteți să creați într-o aplicație clase proprii derivate din `CObject`, iar pentru a utiliza funcționalitatea legată de serializare este suficient să adăugați macrodefiniția `DECLARE_SERIAL` în definiția clasei și macrodefiniția pereche `IMPLEMENT_SERIAL` în implementarea clasei. Acestea macrodefiniții înlocuiesc macrodefinițiile obișnuite, `DECLARE_DYNCREATE` și `IMPLEMENT_DYNCREATE`, folosite în clasele derivate din `CObject`, în scopul adăugării unor elemente de sintaxă necesare unei bune funcționări a serializării.

#### Declararea unei clase serializabile

Dacă aplicația necesită anumite obiecte de date, va trebui să creați clase corespunzătoare care vor implementa și manipula datele respective. Apoi veți adăuga în definițiile de clasă macrodefinițiile pentru serializare și veți implementa salvarea și încărcarea datelor din cadrul obiectelor.

Puteți să creați un fișier de antet (.h) și unul de implementare (.cpp) în care să plasați definiția clasei și, respectiv, codul care o implementează.

#### Crearea unui nou fișier de antet pentru o definiție de clasă

1. Efectuați un clic pe meniul **File** din Developer Studio și selectați opțiunea **New**, așa cum ați proceda pentru crearea unui nou proiect.
2. În cadrul paginii **Files** a casetei de dialog **New** puteți să selectați **C/C++ Header File** pentru a crea un nou fișier.
3. Introduceți numele noului fișier în câmpul **File Name**. (Pentru exemplul nostru, introduceți `blob.h`.)
4. În cazul în care caseta de validare **Add to Project** este selectată (așa cum este în mod implicit), noul fișier de antet va fi adăugat automat la proiect și va apărea într-o fereastră de editare. Asigurați-vă, deci, de selectarea casetei de validare.
5. Efectuați un clic pe **OK** pentru a adăuga în proiect noul fișier de antet și pentru a începe să-l editați.

Listingul 23.1 prezintă o definiție de clasă pentru un obiect serializabil simplu. Clasa este derivată din `CObject`, iar macrodefiniția `DECLARE_SERIAL` satisface necesitățile MFC pentru serializare.

#### LISTINGUL 23.1 LST24\_1.CPP – `blob.h`, definiția clasei pentru obiectul de date `blob` (pată)

```

1 // ** Ne asiguram ca clasa nu este declarata de doua ori
2 #ifndef _BLOB_H
3 #define _BLOB_H
4
5 // ** Derivam clasa CBlob din CObject
6 class CBlob : public CObject
7 {
8 // ** Includem functiile de serializare
9 DECLARE_SERIAL(CBlob); — 1
10
11 public:
12
13 // ** Declaram doi constructori
14 CBlob();
15 CBlob(CPoint ptPosition);
16
17 // ** Declaram o functie de desenare
18 void Draw(CDC* pDC);
19
20 // ** Declaram atributele
21 CPoint m_ptPosition; — 2
22 COLORREF m_crColor;
```

pre. header  
de serializare

(continuare)

## LISTINGUL 23.1 Continuare

```

23 int m_nSize;
24 unsigned m_nShape;
25 };
26
27 #endif // _BLOB_H

```

1. Macrodefiniția `DECLARE_SERIAL` oferă prin expansiune întreg suportul necesar pentru serializare.
2. Atributele necesare pentru desenarea unui obiect blob.

În linia 6, puteți observa că noua clasă `CBlob` este derivată din `CObject`. Atunci când scrieți aplicații compatibile MFC este bine să derivați toate clasele din `CObject`, deoarece aceasta oferă o clasă de bază comună, punând la dispoziție și alte facilități MFC utile, precum informațiile dinamice de tip. Aceste informații ajută MFC să identifice cărui tip de clasă îi aparține un obiect. Informațiile sunt salvate împreună cu datele obiectului, așa că la încărcarea datelor devine posibilă crearea automată a obiectelor instanțe ale claselor corespunzătoare.

Macrodefiniția `DECLARE_SERIAL` din linia 9 adaugă codul necesar pentru funcționarea serializării. În liniile 14 și 15 sunt declarați doi constructori. Primul, cel din linia 14, nu primește nici un parametru, fiind utilizat de infrastructură pentru crearea acestor obiecte odată cu încărcarea datelor corespunzătoare dintr-un fișier de pe disc. Constructorul din linia 15 poate fi utilizat pentru crearea unui obiect și inițializarea automată a membrului poziție pe baza unui `CPoint`. Astfel devine posibilă crearea acestor obiecte pe baza pozițiilor în care se efectuează clic, așa cum veți vedea mai târziu în secțiunea de față.

Declarația `Draw()` din linia 18 corespunde unei funcții de desenare care va fi implementată.

Variabilele din liniile de la 21 la 24 sunt cele supuse serializării la încărcarea sau salvarea obiectului.

### Implementarea clasei serializabile

Odată definită noua clasă, trebuie să scrieți codul de implementare necesar pentru construirea (inițializarea) și manipularea datelor declarate în definiția clasei.

### Crearea unui nou fișier .cpp pentru implementarea clasei

#### Dezvoltarea cu Visual SourceSafe

În cazul în care utilizați o aplicație integrată pentru controlul surselor, așa cum este Visual SourceSafe, veți fi întrebat dacă doriți să adăugați noul fișier sursă în cadrul proiectului SourceSafe (este de preferat). Aplicațiile pentru controlul surselor vă permit să gestionați diferitele versiuni și modificări ale fișierelor sursă, revenind dacă este nevoie la versiuni anterioare. Utilitatea acestor aplicații se dovedește mai ales atunci când mai mulți programatori lucrează la același proiect, caz în care se previne modificarea simultană de către două persoane a aceluiași fișier sursă, ceea ce ar duce la suprascrierea reciprocă a codului.

1. Efectuați un clic pe meniul **F**ile din Developer Studio și selectați opțiunea **N**ew, așa cum ați proceda pentru crearea unui nou proiect.

2. În cadrul paginii **Files** a casetei de dialog **New** puteți să selectați **C/C++ Implementation File** pentru a crea un nou fișier.
3. Introduceți numele noului fișier în câmpul **File Name**. (Pentru exemplul nostru, introduceți `blob.cpp`.)
4. În cazul în care caseta de validare **Add to Project** este selectată (așa cum este în mod implicit), noul fișier de implementare va fi adăugat automat la proiect și va apărea într-o fereastră de editare. Asigurați-vă, deci, de selectarea casetei de validare.
5. Efectuați un clic pe **OK** pentru a adăuga în proiect noul fișier de implementare și a începe să-l editați.

O clasă serializabilă trebuie să aibă un constructor generic – adică, un constructor fără parametri. Acesta este necesar pentru crearea obiectelor în cadrul procesului de serializare dintr-un fișier de pe disc. În consecință, MFC va avea la dispoziție un constructor standard pentru toate tipurile de obiecte, putând să le creeze la încărcarea unui fișier. Prin implementarea unui constructor fără parametri devine posibilă crearea oricărui obiect aparținând unei clase serializabile, inițializarea variabilelor membru ale noului obiect revenind implementării funcției `Serialize()`.

Puteți, de asemenea, să adăugați oricâți alți constructori care să vă permită crearea de noi instanțe de obiect, inițializate cu orice date doriți.

Fișierul de implementare al oricărei clase serializabile trebuie să conțină macrodefiniția `IMPLEMENT_SERIAL`. Această macrodefiniție primește trei parametri. Primul este chiar numele clasei în cauză, al doilea este clasa de bază, iar cel de-al treilea (`wschema`) este o valoare `UINT` care reprezintă un număr de versiune. Dacă adăugați în clasă noi variabile membru pe care doriți să le serializați, puteți să incrementați numărul de versiune și, eventual, să-l verificați pentru a păstra compatibilitatea cu versiunile mai vechi de fișiere arhivă serializate care există.

#### Controlul versiunilor

Dacă scrieți aplicații comerciale care folosesc serializarea, este foarte important să mențineți o schemă de versiune pentru diferitele versiuni de fișiere/obiecte și fie să converțiți, fie să păstrați compatibilitatea cu obiectele mai vechi. În caz contrar, v-ați putea găsi în neplăcuta situație de a face inutilizabile toate datele de pe disc ale unui client în momentul în care îi furnizați o nouă versiune a aplicației.

Implementarea pentru clasa `CBlob` este prezentată în listingul 23.2. Acest exemplu de implementare conține macrodefiniția `IMPLEMENT_SERIAL` și funcțiile constructor, precum și o funcție responsabilă de reprezentarea obiectului.

### LISTINGUL 23.2 LST24\_2.CPP – blob.cpp, implementarea clasei pentru obiectul de date Blob

```

1 // ** includem fisierul de antet standard
2 #include "stdafx.h"
3 #include "blob.h"
4

```

(continuare)

## LISTINGUL 23.2 Continuare

```

5 // ** Adaugam declaratia de implementare pentru serializare
6 IMPLEMENT_SERIAL(CBlob, CObject, 1)
7
8 // ** Implementarea constructorului implicit
9 CBlob::CBlob()
10 {
11 }
12
13 // ** Implementarea constructorului pe baza pozitiei
14 CBlob::CBlob(CPoint ptPosition) — ❶
15 {
16 // ** Initializam generatorul de numere aleatoare
17 srand(GetTickCount());
18
19 // ** Retinem pozitia specificata
20 m_ptPosition = ptPosition;
21
22 // ** Initializam celelalte attribute cu valori aleatoare
23 m_crColor = RGB(rand() % 255, rand() % 255, rand() % 255);
24 m_nSize = 10 + rand() % 30;
25 m_nShape = rand();
26 }
27
28 void CBlob::Draw(CDC* pDC) — ❷
29 {
30 // ** Cream si selectam o pensula colorata
31 CBrush brDraw(m_crColor);
32 CBrush* pOldBrush = pDC->SelectObject(&brDraw);
33 CPen* pOldPen =
34 (CPen*)pDC->SelectStockObject(NULL_PEN);
35 // ** Initializam generatorul de numere aleatoare pe baza formei
36 srand(m_nShape);
37 for(int n=0; n<3; n++)
38 {
39 // ** Stabilim pozitia petei si adunam un deplasament
40 // aleator
41 CPoint ptBlob(m_ptPosition);
42 ptBlob += CPoint(rand() % m_nSize, rand() % m_nSize);
43
44 // ** Cream un dreptunghi si trasam o elipsa
45 CRect rcBlob(ptBlob, ptBlob);
46 rcBlob.InflateRect(m_nSize, m_nSize);
47 pDC->Ellipse(rcBlob);
48 }

```

```

49 // ** Selectam la loc obiectele GDI originale
50 pDC->SelectObject(pOldBrush);
51 pDC->SelectObject(pOldPen);
52 }

```

❶ Acest constructor poate fi folosit la crearea unui obiect blob pe baza poziției în care s-a efectuat clic, restul atributelor fiind stabilite aleator.

❷ Funcția Draw() permite unui obiect Blob să se reproducă în cadrul unui context dispozitiv transmis ca pointer din funcția OnDraw() a reprezentării.

Directivele #include din liniile 2 și 3 informează compilatorul despre definiția clasei și despre definițiile MFC. Macrodefiniția IMPLEMENT\_SERIAL() din linia 6 corespunde macrodefiniției DECLARE\_SERIAL() din definiția clasei.

## Utilizarea machetelor de vectori pentru asigurarea stricteții tipurilor

COBArray nu asigură strictețea tipurilor, ceea ce înseamnă că acceptă și stochează fără probleme orice obiect derivat din CObject. Sunt cazuri în care este exact ceea ce doriți, dar alții ați putea avea nevoie de un vector în care să puteți stoca un anumit tip de obiect. Mai multe machete, așa cum este CTypedPtrArray, vă ajută să asigurați strictețea tipurilor. Încercarea de adăuga un obiect de alt tip într-un vector de obiecte cu tip bine determinat va duce la generarea unui mesaj de avertizare în legătură cu încălcarea unei aserțiuni.

Constructorul implicit este declarat în linia 9. La încărcarea unui fișier, mecanismul de serializare creează obiectele necesare prin intermediul constructorului implicit, după care transmite acestor instanțe de obiecte datele citite din fișier.

Începând cu linia 14 este implementat un constructor specific, care permite crearea acestor obiecte pe baza pozițiilor în care se efectuează clic, restul atributelor având valori aleatoare. Funcția Draw() din linia 28 răspunde de desenarea obiectelor în cadrul unui context dispozitiv al reprezentării, transmis ca pointer.

## Stocarea datelor în cadrul documentului

Odată definite clasele aplicației, trebuie să creați instanțe ale acestor clase care să reprezinte datele utilizatorului ca obiecte individuale. Atunci când creați astfel de obiecte se impune gestionarea lor adecvată, astfel încât să puteți să manipulați datele acestora, să le salvați și să le încărcăți și, în cele din urmă, să distrugeți obiectele. De fiecare dată când încărcăți sau salvați datele unui document trebuie să parcurgeți toate obiectele de date și să le serializați pe fiecare în parte.

Unele clase MFC ajută la gestionarea acestor obiecte și suportă serializarea, iterând automat instanțele obiectelor gestionate. O astfel de clasă este COBArray. COBArray este una dintre clasele colecție din MFC. Ea reprezintă un vector extensibil care poate reține obiecte derivate din CObject, deci și obiectele de date pe care le definiți. Mai mult, suportă serializarea și vă va ajuta la serializarea datelor documentului.

Ați putea să adăugați un obiect COBArray în definiția clasei document CPersistDoc, după comentariul // Attributes și specificatorul de acces public:, ca aici:

```
// Attributes
public:
 CObArray m_BlobArray;
```

Acest nou obiect `m_BlobArray` va reține și va gestiona obiecte derivate din `CObject` (așa cum sunt cele de tip `CBlob`).

Pe lângă păstrarea obiectelor de date, documentul trebuie să asigure și distrugerea lor la încheierea aplicației. Această operație se poate implementa printr-o nouă funcție, prezentată în listingul 23.3, adăugând în definiția clasei declarația de funcție corespunzătoare:

```
void DeleteBlobs();
```

### LISTINGUL 23.3 LST24\_3.CPP – Implementarea și utilizarea funcției `DeleteBlobs()` în scopul distrugerii tuturor obiectelor blob alocate

```
1 void CPersistDoc::DeleteBlobs() — ❶
2 {
3 // ** Stergem obiectele blob alocate
4 for(int i=0; i<m_BlobArray.GetSize(); i++)
5 delete m_BlobArray.GetAt(i);
6 m_BlobArray.RemoveAll();
7 }
8
9 CPersistDoc::~CPersistDoc()
10 {
11 DeleteBlobs();
12 }
13
14 BOOL CPersistDoc::OnNewDocument()
15 {
16 if (!CDocument::OnNewDocument())
17 return FALSE;
18
19 // TODO: add reinitialization code here
20 // (SDI documents will reuse this document)
21
22 DeleteBlobs(); — ❷
23
24 return TRUE;
25
```

❶ Funcția `DeleteBlobs()` parcurge toate obiectele blob, eliberând memoria alocată pentru fiecare dintre ele, după care șterge și vectorul propriu-zis.

❷ La crearea unui nou document trebuie distruse vechile obiecte blob, deoarece infrastructura SDI refolosește același obiect document.

Implementarea noii funcții este prezentată în listingul 23.3 (liniile de la 1 la 7). Funcția `DeleteBlobs()` parcurge vectorul `CObArray` de obiecte `CBlob` și le șterge pe fiecare dintre acestea. Apoi este șters și vectorul, prin apelul funcției `RemoveAll()`.

Atunci când documentul este distrus, se apelează funcția destructor a clasei, iar aici ar trebui să apelezi funcția nou creată pentru a șterge obiectele de date existente, ca în linia 11. Este posibil ca utilizatorul să creeze un nou document în timp ce există un document deschis; în acest caz, infrastructura refolosește documentul existent. Într-o astfel de situație este necesară ștergerea obiectelor de date reținute curent în cadrul documentului. Aceasta se poate face în funcția `CPersistDoc::OnNewDocument()`, apelând `DeleteBlobs()` ca în linia 22.

Ne-am ocupat de gestionarea și ștergerea obiectelor de date, dar avem nevoie de un mecanism care să permită utilizatorului să creeze noi obiecte și să le insereze în cadrul documentului. Acest mecanism este foarte probabil să difere de la o aplicație la alta, depinzând de implementarea interfeței cu utilizatorul. Odată creat un nou obiect de date, acesta poate fi inserat într-un container, așa cum este `CObArray`. Această clasă oferă o funcție `Add()` care primește un singur parametru, un pointer la noul obiect.

### Utilizarea clasei `CObArray`

Ca alternativă la adăugarea obiectelor prin intermediul funcției membru `Add()`, `CObArray` vă permite totodată să inserați obiecte într-o anumită poziție, oferindu-vă funcția membru `InsertAt()` (căreia îi transmiteți poziția în care vreți să fie inserat obiectul și un pointer la obiectul de inserat). Există și o funcție corespunzătoare, `RemoveAt()`, care permite ștergerea unui obiect aflat într-o poziție anume. Funcțiile `SetAt()` și `GetAt()` pot fi utilizate pentru a modifica sau extrage elemente existente ale vectorului pe baza indicelui asociat.

Metoda cea mai simplă de a crea aceste obiecte în cadrul programului este să permiteți utilizatorului să efectueze clic în fereastra reprezentării, construind un obiect `CBlob` pe baza poziției în care s-a efectuat clicul. În acest scop veți avea nevoie de o funcție de tratare a mesajului corespunzător, `OnLButtonDown()`. Această funcție poate fi creată cu ajutorul `ClassWizard`, așa cum o ilustrează secvența de pași „Adăugarea unei funcții de tratare” din capitolul 19, „Utilizarea reprezentărilor de tip listă, arbore, editare formatată și HTML”.

Adăugați liniile de mai jos în noua funcție de tratare `OnLButtonDown()` (după comentariile `// TODO`) pentru a crea un nou obiect blob pe baza parametrului `point`, adăugându-l în vectorul `m_BlobArray` din cadrul documentului:

```
GetDocument()->m_BlobArray.Add(new CBlob(point));
Invalidate();
```

Funcția `Invalidate()` va determina actualizarea reprezentării și, implicit, desenarea obiectelor, prin intermediul codului prezentat în listingul 23.4.

### LISTINGUL 23.4 LST24\_4.CPP – Afișarea obiectelor blob în `OnDraw()`

```
1 void CPersistView::OnDraw(CDC* pDC)
2 {
3 CPersistDoc* pDoc = GetDocument();
```

(continuare)

## LISTINGUL 23.4 Continuare

```

4 ASSERT_VALID(pDoc);
5
6 // TODO: add draw code for native data here
7
8 for(int i=0;i<pDoc->m_BlobArray.GetSize();i++) — ❶
9 {
10 CBlob* pBlob=(CBlob*)pDoc->m_BlobArray.GetAt(i);
11 pBlob->Draw(pDC);
12 }
13 }

```

❶ Reprezentarea se rezumă la a parcurge vectorul de obiecte blob, solicitând fiecărui obiect să-și genereze propria reprezentare.

În listingul 23.4, obiectele sunt iterate între liniile 8 și 12, apelându-se funcția `Draw()` a fiecărui obiect blob, ca în linia 11.

Compilatorul va avea nevoie de definiția de clasă a acestor obiecte:

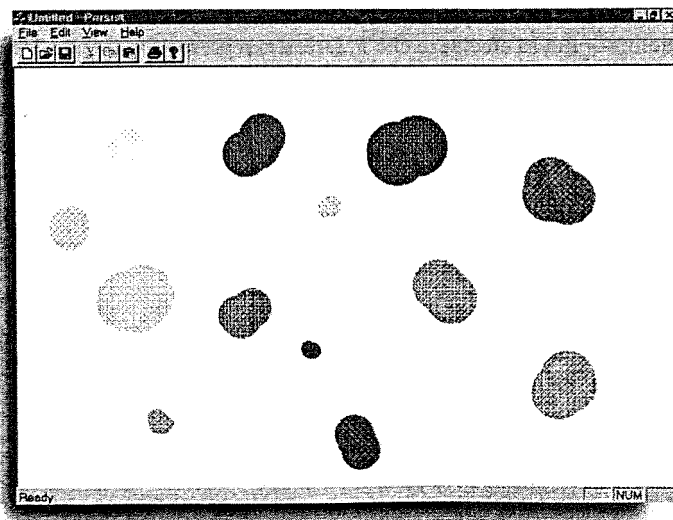
```
#include "blob.h"
```

Adăugați această linie în fișierul `PersistView.cpp`, după celelalte directive `#include`.

Dacă asamblați și rulați aplicația `Persist`, veți putea să efectuați clic în fereastra reprezentării și să adăugați astfel noi obiecte blob, așa cum o înfățișează figura 23.2.

FIGURA 23.2

Reprezentarea obiectelor blob în cadrul aplicației `Persist`.



## VEDEȚI...

- Pentru a afla mai multe despre documente și despre datele din cadrul acestora, vedeți pagina 260.
- Pentru a afla despre desenarea de figuri umplute în cadrul contextelor dispozitiv, vedeți pagina 372.

## Serializarea obiectelor de date

Pentru a efectua serializarea obiectelor de date va trebui să implementați o funcție `Serialize()` corespunzătoare, cu scopul de a transforma toate variabilele membru într-o listă secvențială care să poată fi salvată sau încărcată dintr-un fișier. Iată definiția funcției virtuale `Serialize()`:

```
// ** Supradefinirea funcției Serialize
virtual void Serialize(CArchive& ar);
```

Declarația de mai sus a funcției `Serialize()` trebuie adăugată în definiția de clasă din fișierul de antet `.h` corespunzător (de exemplu, `Blob.h`). Apoi trebuie adăugată o implementare a funcției în cadrul fișierului de implementare `.cpp` al clasei (de exemplu, `Blob.cpp`).

## Adăugarea în propriile clase a funcțiilor virtuale

Dacă ați folosit anterior caseta de dialog `New Virtual Override` cu scopul de a adăuga funcții virtuale prin intermediul meniului contextual din pagina `ClassView`, veți observa că această facilități nu poate fi utilizată pentru adăugarea de funcții virtuale în propriile clase. În schimb, pentru o astfel de funcție va trebui să adăugați manual declarația și codul de implementare în fișierele de definiție (`.h`) și de implementare (`.cpp`) ale clasei, cu ajutorul editorului de text.

Fișierul de pe disc este încapsulat în obiectul `CArchive` transmis funcției `Serialize()` de către infrastructură.

Obiectul `CArchive` ajută la transformarea datelor într-o listă și reține direcția transferului, fie în sensul stocării, fie în sensul încărcării. În momentul în care utilizatorul optează pentru încărcarea sau salvarea unui document, infrastructura creează automat un obiect arhivă și deschide fișierul, gata pregătit pentru transfer. Apoi apelează funcția `Serialize()` a documentului, transmitându-i obiectul arhivă.

`CArchive` oferă două funcții sub forma operatorilor `<<` și `>>`. Aceștia sunt folosiți la transferul datelor înspre sau dinspre arhivă, salvarea unei valori efectuându-se astfel:

```
ar << m_MyVar
```

Încărcarea unei valori dintr-un fișier se face astfel:

```
ar >> m_MyVar
```

Atunci când scrieți funcția `Serialize()` a unui obiect, puteți să determinați dacă obiectul este salvat sau încărcat prin intermediul funcțiilor membru `IsLoading()` și `IsStoring()` ale obiectului `CArchive`. Apoi veți avea, de regulă, câteva linii care folosesc operatorii `<<` și `>>` pentru a salva și, respectiv, încărca fiecare dintre variabilele membru supuse serializării. Toate acestea sunt ilustrate în cadrul obiectului `CBlob`, în listingul 23.5.

**LISTINGUL 23.5 LST24\_5.CPP – Implementarea serializării în cadrul clasei CBlob**

```

1 void CBlob::Serialize(CArchive& ar)
2 {
3 CObject::Serialize(ar);
4 if (ar.IsStoring())
5 {
6 ar << m_ptPosition; — ❶
7 ar << m_crColor;
8 ar << m_nSize;
9 ar << m_nShape;
10 }
11 else
12 {
13 ar >> m_ptPosition; — ❷
14 ar >> m_crColor;
15 ar >> m_nSize;
16 ar >> m_nShape;
17 }
18 }

```

❶ La salvare, variabilele membru sunt înscrise în arhivă.

❷ La încărcare, arhiva este desfăcută și distribuită în cadrul variabilelor membru.

În linia 4 a listingului 23.5 este apelat membrul `IsStoring()` al clasei `CArchive`, determinându-se astfel dacă urmează o operație de salvare sau de încărcare a datelor. În cazul unei operații de salvare trebuie să înscriseți datele în cadrul arhivei. În cazul unei operații de încărcare trebuie să preluați datele din arhivă. Liniile de la 6 la 9 se ocupă de salvarea datelor în arhivă. Operatorul `<<` transmite arhivei fiecare atribut membru al clasei blob, acesta fiind înscris mai departe în fișierul de pe disc.

Liniile de la 13 la 16 efectuează operația opusă, iar operatorul `>>` primește o variabilă membru și depune în cadrul său datele preluate din arhivă.

Pentru salvarea sau încărcarea obiectelor documentului trebuie să apălați funcția `Serialize()` a fiecărui obiect. Aceasta se poate face prin modificarea implementării `Serialize()` din cadrul documentului, apelând de aici funcția `Serialize()` pentru fiecare obiect.

Păstrarea obiectelor de date într-un obiect `COBArray`, așa cum este membrul `m_BlobArray` din exemplul nostru, face lucrurile mult mai ușoare. `COBArray` oferă suport pentru serializare, fiind capabil să parcurgă automat toate obiectele pe care le conține și să le salveze. Similar, la încărcarea unui document aflat pe disc, va determina numărul de obiecte și tipul lor, le va crea automat (cu ajutorul constructorului implicit) și apoi va apăla funcția `Serialize()` a fiecăruia dintre ele în scopul încărcării. Prin urmare, tot ce aveți de făcut este un singur apel `Serialize()`, efectuat asupra vectorului care conține obiectele, ca aici:

```

void CPersistDoc::Serialize(CArchive& ar)
{
 m_BlobArray.Serialize(ar);
}

```

**Serializarea datelor documentului**

Majoritatea aplicațiilor vor stoca, evident, date mult mai sofisticate decât cele din exemplul nostru. Oricum, funcția `Serialize()` a documentului reprezintă un punct de start pentru procesul de serializare a datelor aplicației. Probabil că pentru încărcarea și salvarea tuturor datelor va trebui să adăugați în funcția `Serialize()` și alți vectori, precum și obiecte colecție mult mai sofisticate.

Operați aceste modificări în cadrul programului, după care asamblați și rulați aplicația pentru a vedea cum codul de serializare încarcă și salvează obiectele blob. Paragrafele următoare vă arată cum puteți să salvați și să încărcați obiectele blob în cadrul aplicației create.

Desenați câteva obiecte blob efectuând clicuri în interiorul ferestrei. Deschideți meniul **File** și selectați **Save As**. Infrastructura afișează automat caseta de dialog standard **Save As**, permițându-vă să înlocuiți numele de fișier implicit, `Untitled.blb`, cu numele dorit pentru fișierul document. Puteți să modificați numele de fișier, iar în momentul în care efectuați un clic pe **Save** se declanșează operația de serializare, toate obiectele blob fiind salvate pe disc.

Pentru a vă convinge, închideți aplicația, lansați-o din nou și selectați opțiunea **Open** a meniului **File**. Datorită filtrului `.blb` pe care l-ați specificat în **AppWizard**, în mod implicit veți vedea numai fișierele cu obiecte blob, așa că fișierul pe care l-ați salvat ar trebui să fie singurul afișat. Selectați-l și efectuați un clic pe butonul **Open** pentru a-l încărca. Iată! – ar trebui să vedeți obiectele blob create, aflate în pozițiile lor originale. Acum puteți să creați și alte obiecte blob și să salvați din nou documentul sau puteți să creați un nou document și să-l salvați sub un alt nume, una peste alta fiind posibile toate operațiile tipice cu fișiere pe care le permit majoritatea aplicațiilor. Puteți să stingeți calculatorul și să plecați două săptămâni, iar fișierul cu obiecte blob va rămâne acolo, cu aceleași obiecte blob aflate în aceleași poziții și având aceleași forme și culori.

**Utilizarea listei cu fișierele cel mai recent utilizate**

O altă facilitare pe care infrastructura o pune la dispoziție în mod gratuit este lista cu fișierele cel mai recent utilizate. Dacă salvați mai multe fișiere diferite cu obiecte blob, veți putea să observați că în meniul **File** sunt afișate fișierele folosite cel mai recent. Aceste fișiere pot fi deschise repede și ușor, selectându-le direct din meniu.

**Utilizarea clasei CRecentFileList**

Infrastructurile standard ale aplicațiilor SDI și MDI folosesc clasa `CRecentFileList` pentru a implementa lista cu fișierele utilizate recent. Puteți să folosiți această clasă și direct (nu neapărat prin intermediul infrastructurii) pentru a implementa propriile liste cu fișiere recent utilizate, atașându-le la meniurile dorite. Clasa `CRecentFileList` vă permite să specificați pentru fiecare fișier numele complet al acestuia și o etichetă de afișat. Este posibilă, de asemenea, actualizarea automată a unui meniu în scopul adăugării sau eliminării de fișiere din listă.



## Înregistrarea tipurilor de documente

Probabil că ați văzut cum simpla efectuare a unui dublu clic pe un fișier document care are asociat un anumit tip poate să lanseze aplicații Windows. Acestea sunt fișiere ale căror tipuri sunt înregistrate; prin simpla adăugare în propria aplicație a două linii de cod puteți stabili același gen de asociere. Următoarele două linii trebuie adăugate în funcția membru `InitInstance()` a clasei aplicație, imediat după linia `AddDocTemplate(pDocTemplate);`:

```
RegisterShellFileTypes();
EnableShellOpen();
```

### VEDEȚI...

➤ Pentru a afla mai multe despre documente și despre datele acestora, vedeți pagina 246.

## Manipularea fișierelor

Serializarea documentelor este foarte utilă în cadrul aplicațiilor SDI și MDI care trebuie să stocheze datele din cadrul documentelor, dar sunt situații în care va trebui să creați, să citiți și să scrieți fișiere în mod direct.

MFC oferă o clasă de încapsulare pentru fișierele de pe disc, clasă numită `CFile`. Această clasă încorporează toate operațiile cu fișiere și toate atributele asociate unui fișier de pe disc. Prin crearea și utilizarea obiectelor `CFile` puteți să creați, să deschideți, să citiți și să scrieți fișiere, apelând la funcțiile oferite de clasa `CFile`.

Fie că serializați datele unui document, fie că scrieți direct într-un fișier, veți folosi până la urmă clasa `CFile` sau una dintre derivatele sale. Clasa `CArchive`, care se folosește la serializare, intră de fapt în legătură cu un obiect `CFile` care mijlocește citirea sau scrierea datelor documentului. Secțiunea de față prezintă modul de operare al clasei `CFile` și se oprește asupra unor versiuni mai specializate ale acestei clase.

## Utilizarea clasei `CFile`

Un obiect `CFile` poate fi construit în trei feluri. Constructorul cel mai simplu nu primește nici un parametru și se rezumă la a crea un obiect fișier care nu este deschis, urmând ca ulterior să apelați funcția `Open()` a acestuia pentru a crea sau a deschide un fișier real de pe disc. O a doua versiune a constructorului primește un parametru, un indicator `hFile`, care trebuie să corespundă unui fișier deschis. Această versiune poate fi utilizată în scopul atașării unui obiect `CFile` la un fișier deja deschis. A treia versiune de constructor necesită doi parametri: primul este un șir care conține calea și numele fișierului pe care doriți să-l deschideți sau să-l creați, iar cel de-al doilea reprezintă o combinație de indicatori de deschidere a fișierului.

### Fișierele și sistemul de operare

Asemeni multor clase MFC, `CFile` încapsulează un indicator de fișier folosit de nucleul Win32 al sistemului de operare. Acest indicator poate fi accesat direct, aflându-se în variabila membru `m_hFile` a obiectului `CFile`. În cazul în care nu a fost deschis sau creat nici un fișier valid, această variabilă va avea valoarea `CFile::hFileNull`, iar în caz contrar va reține indicatorul asociat fișierului curent.

## Deschiderea fișierelor

Una dintre cele mai importante operații în gestionarea fișierelor este deschiderea inițială a fișierului. Prin specificarea unor diferiți indicatori puteți să anunțați anticipat modul în care vreți să lucrați cu fișierul specificat. Ați putea dori să creați un nou fișier, să deschideți un fișier exclusiv pentru citire sau să deschideți un fișier atât pentru citire, cât și pentru scriere. Funcția `Open()` vă permite să precizați o gamă largă de moduri de acces la un fișier, transmitând prin cel de-al doilea parametru, `nOpenFlags`, oricare dintre indicatorii prezentați în tabelul 23.1. Primul parametru reprezintă, în acest caz, numele fișierului pe care doriți să-l deschideți sau să-l creați. Unii indicatori pot fi combinați, un exemplu fiind cel de mai jos:

```
CFile fileMyFile;
fileMyFile.Open("MyFile.txt",
 CFile::modeCreate + CFile::modeNoTruncate);
```

Indicatorii specificați aici vor avea ca efect deschiderea fișierului numit `MyFile.txt` în cazul în care acesta există; altfel, va fi creat un nou fișier, gol, având același nume. Dacă ați fi precizat doar indicatorul `CFile::modeCreate`, s-ar fi creat un nou fișier chiar dacă exista deja un altul cu numele indicat, deci indicatorul `CFile::modeNoTruncate` alterează efectul primului indicator astfel încât fișierele existente să nu fie reduse la lungime zero. Pentru combinarea indicatorilor am apelat la operatorul `+`, dar de multe ori veți întâlni operatorul SAU logic, `|`, folosit ca aici:

```
fileMyFile.Open("MyFile.txt",
 CFile::modeCreate | CFile::modeNoTruncate);
```

Este o formulă la fel de corectă, deoarece acești indicatori sunt măști de biți, iar operatorul SAU logic îi combină la fel ca și adunarea aritmetică.

Dacă fișierul este deschis cu succes, funcția `Open()` întoarce valoarea `TRUE`; altfel, se întoarce valoarea `FALSE` pentru a indica faptul că deschiderea fișierului a eșuat. La deschidere există posibilitatea opțională de a transmite un al treilea parametru, un pointer la un obiect `CFileException`, iar în cazul în care deschiderea fișierului nu reușește, acest obiect va conține detalii indicând cauza eșecului.

### Utilizarea obiectului `CFileException`

Dacă la deschiderea unui fișier apare o eroare, este transmis un pointer la un obiect `CFileException`. Cauza erorii poate fi determinată pe baza variabilelor `m_cause` (sub forma unui element dintr-o enumerare definită în `CFileException`) și `m_IOException` (sub forma unui cod de eroare specific sistemului de operare), ambele fiind membre ale obiectului `CFileException`. Spre exemplu, valori tipice pentru `m_cause` sunt `CFileException::fileNotFound`, `CFileException::accessDenied` și `CFileException::diskFull`.



**TABELUL 23.1 Indicatori pentru modul de deschidere care pot fi transmiși, eventual în combinație, funcției CFile::Open()**

| Indicator             | Descriere                                                                                                                                                                        |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CFile::modeCreate     | Creează întotdeauna un nou fișier (chiar și dacă există un fișier cu numele specificat).                                                                                         |
| CFile::modeNoTruncate | Acest indicator poate fi combinat cu CFile::modeCreate, efectul fiind crearea unui nou fișier dacă nu există un fișier cu numele specificat sau deschiderea fișierului existent. |
| CFile::modeRead       | Fișierul va fi deschis exclusiv pentru citire; nu este permisă scrierea datelor în fișier.                                                                                       |
| CFile::modeWrite      | Fișierul va fi deschis exclusiv pentru scriere; nu este permisă citirea datelor din fișier.                                                                                      |
| CFile::modeReadWrite  | Fișierul este deschis atât pentru citire, cât și pentru scriere.                                                                                                                 |
| CFile::shareDenyNone  | Fișierul poate fi citit și scris de către alte procese.                                                                                                                          |
| CFile::shareExclusive | Nici un alt proces nu poate să citească sau să scrie fișierul cât timp acesta este deschis în procesul curent.                                                                   |
| CFile::shareDenyRead  | Nici un alt proces nu poate să citească fișierul cât timp acesta este deschis.                                                                                                   |
| CFile::shareDenyWrite | Nici un alt proces nu poate să scrie în fișier cât timp acesta este deschis.                                                                                                     |
| CFile::typeText       | Folosit pentru prelucrări specifice conținutului textual în cadrul unora dintre clasele derivate.                                                                                |
| CFile::typeBinary     | Nu se efectuează nici un fel de prelucrări speciale asupra caracterelor la citirea sau scrierea acestora în fișier. Acest indicator este necesar doar în unele clase derivate.   |

## Citirea și scrierea unui fișier

Odată deschis un fișier, puteți să efectuați operații specifice de citire și de scriere în măsura permisă de modul curent de acces. Pentru efectuarea acestor operații, clasa CFile oferă funcțiile Read() și Write(). La operațiile de citire și de scriere există asociată o poziție curentă în fișier. După deschiderea fișierului, această poziție este stabilită la începutul acestuia. Dacă citiți 200 de octeți din fișier, poziția va fi actualizată, iar următorul apel Read() va citi în continuarea primilor 200 de octeți. Secțiunea „Manipularea poziției curente în fișier”, prezentată mai târziu în capitolul de față, vă arată cum puteți să modificați această poziție curentă. Un apel tipic al funcției Read() ar putea arăta astfel:

```
CFile myFile("MyFile.txt", CFile::modeRead);
char arMyReadBuffer[200];
UINT uBytesRead =
 myFile.Read(arMyReadBuffer, sizeof(arMyReadBuffer));
```

Funcția Read() primește doi parametri. Primul dintre aceștia reprezintă adresa unui buffer destinație. Orice date citite vor fi depuse în acest buffer. Al doilea parametru specifică

funcției Read() câți octeți doriți să citiți; pot fi doar câțiva octeți sau poate fi întreaga dimensiune a bufferului, ca în exemplul anterior.

### Depășirea bufferului la citire

Trebuie să aveți grijă să nu efectuați niciodată o operație Read() care să citească mai mulți octeți decât încap în buffer. Altfel, datele citite pot să suprascrie zone vitale ale programului, fiind o cale sigură de provocare a unei blocări sau a unor efecte imprevizibile.

La încheierea sa, funcția Read() întoarce numărul de octeți citiți. Acesta poate fi egal cu numărul de octeți solicitat sau, dacă a fost atins sfârșitul fișierului, poate fi mai mic, indicând că au fost citiți toți octeții rămași din fișier. Dacă este întoarsă valoarea zero, înseamnă că nu au mai rămas octeți de citit din fișier.

Puteți pune în practică toate aceste informații pentru a crea o simplă aplicație editor de tip dialog. Adăugând codul corespunzător în cadrul funcției OnInitDialog() a casetei de dialog, ați putea să citiți un fișier text de pe disc, să-l editați și apoi să-l salvați la apăsarea butonului OK.

Creați o aplicație de tip dialog numită FileEdit, urmând pașii prezentați în capitolul 5, „Utilizarea controalelor textuale”, și apoi adăugați în caseta de dialog principală un control de editare mare, având selectat stilul **MultiLine**. În continuare, puteți să apelați la ClassWizard pentru a mapa peste controlul de editare o variabilă de tip CString, numită m\_EditBox (operație ilustrată, de asemenea, în capitolul 5).

Adăugați în funcția OnInitDialog() a clasei CFileEditDlg liniile de cod prezentate în listingul 23.6 pentru a citi conținutul fișierului text invariabil numit C:\MyFile.txt.

### LISTINGUL 23.6 LST24\_6.CPP – Citirea conținutului unui fișier în cadrul unui control de editare

```
1 // TODO: Add extra initialization here
2
3 // Declaram si deschidem pentru citire un obiect fisier
4 CFile fileEditText;
5 if(fileEditText.Open("C:\\MyFile.txt", CFile::modeRead))
6 {
7 // Cream un buffer mare in care vom citi textul
8 char cBuf[512];
9 UINT uBytesRead;
10
11 // Continuum sa citim pana cand nu mai avem ce
12 while(uBytesRead = 1
13 fileEditText.Read(cBuf, sizeof(cBuf)-1))
14 {
15 // Adaugam caracterul NULL la sfarsitul sirului
16 cBuf[uBytesRead] = NULL;
17 }
```

(continuare)

## LISTINGUL 23.6 Continuare

```

18 // Atasam bufferul la variabila CString mapata
19 m_EditBox += CString(cBuf);
20 }
21
22 // Inchidem fisierul
23 fileEditText.Close();
24
25 // Inscriem sirul m_EditBox in cadrul controlului de editare
26 UpdateData(FALSE);
27 }

```

- ❶ Bucla while se execută până când funcția Read() întoarce o valoare care arată că nu a mai citit nici un octet; acesta este sfârșitul de fișier.

## Utilizarea caracterului special backslash

Observați că numele de fișier din linia 5 include o secvență de două caractere backslash. Nu este o greșeală de redactare; backslash este un caracter special de prefixare, care vă permite specificarea unor coduri precum `\n`, `\r` sau `\b`, pentru caracterele linie nouă, retur de car sau backspace. Dacă doriți să specificați caracterul backslash, trebuie să introduceți o secvență `\\` pentru fiecare caracter `\` dorit.

În listingul 23.6, `fileEditText` este un obiect de tip `CFile`, creat în linia 4 și deschis în linia 5 pentru fișierul cu numele predefinit `C:\MyFile.txt`.

Bufferul declarat în linia 8 este utilizat în apelul `Read()` din linia 13 pentru stocarea octeților citiți din fișier. Al doilea parametru transmis precizează că numărul de octeți solicitat este cu unu mai puțin decât dimensiunea bufferului. Aceasta pentru a putea adăuga un caracter `NULL` la sfârșitul datelor citite din fișier, rezultând astfel un *șir cu terminator nul*. Odată adăugat terminatorul nul, bufferul poate fi convertit la un obiect `CString` și atașat, în linia 19, la sfârșitul șirului `m_EditBox`.

## Șiruri cu terminator nul

Un șir cu terminator nul este un vector de caractere în care secvența caracterelor semnificative este urmată de caracterul `NULL` (sau zero), acesta indicând sfârșitul șirului în cadrul întregului vector. Șirurile cu terminator nul sunt uzuale în codul C și există mai multe funcții care operează cu acestea. De exemplu, `strlen()` întoarce lungimea unui șir. Funcția `strcpy()` copiază șirul sursă precizat ca al doilea parametru în cadrul primului parametru. Funcția `strcmp()` compară două șiruri și întoarce zero dacă sunt identice. Multe altele astfel de funcții pot fi găsite în documentația online oferită de Microsoft, sub titlul *String-Manipulation Routines*. Pentru utilizarea acestor funcții este necesară includerea fișierului de antet `string.h`.

Dacă fișierul depășește la prima iterație dimensiunea bufferului, bucla se execută din nou, citind astfel în mod repetat până când se epuizează conținutul fișierului. În momentul în care `uBytesRead` devine zero în urma operației de citire, bucla `while` se încheie și fișierul este închis prin apelul funcției `Close()` din linia 23. Controlul de editare este actualizat pe baza conținutului variabilei `m_EditBox` asociate, prin apelul `UpdateData(FALSE)` din linia 26.

Cu aceasta am încheiat partea de citire a editorului de fișier. Dacă asamblați și rulați programul după adăugarea acestor linii de cod, veți vedea că este afișat textul aflat în fișierul `C:\MyFile.txt`. Puteți să creați acest fișier rulând programul `NotePad` din `Windows`, scriind ceva și apoi salvând textul sub numele `C:\MyFile.txt`.

După ce utilizatorul apasă **OK** în urma editării textului, trebuie să înregistrați modificările în fișier. Aceasta se poate face forțând crearea unui nou fișier (cu suprascrierea oricăror date existente) și apelând funcția `Write()` pentru a depune înapoi în fișier conținutul obiectului `CString` mapat pe controlul de editare.

Folosiiți `ClassWizard` pentru a adăuga o funcție de tratare a mesajului `BN_CLICKED` generat de butonul **OK**; această funcție se va numi `OnOK()` și în cadrul său veți implementa codul care salvează conținutul casetei de editare, cod prezentat în listingul 23.7. Dacă aveți nevoie de asistență în ceea ce privește adăugarea funcției de tratare, consultați secvența de pași „Adăugarea unei funcții de tratare a clicului asupra unui buton” din capitolul 4, „Utilizarea controalelor buton”.

## LISTINGUL 23.7 LST24\_7.CPP – Înscrierea într-un fișier de pe disc a datelor din cadrul controlului de editare

```

1 void CFileEditDlg::OnOK()
2 {
3 // TODO: Add extra validation here
4
5 // Actualizam sirul m_EditBox pe baza controlului de editare
6 UpdateData(TRUE);
7
8 // Declaram si deschidem pentru citire un obiect fisier
9 CFile fileEditText;
10 if (fileEditText.Open("C:\\MyFile.txt",
11 CFile::modeCreate + CFile::modeWrite))
12 {
13 // Scriem intreg sirul
14 fileEditText.Write(
15 (LPCTSTR)m_EditBox,m_EditBox.GetLength()); ❶
16
17 // Inchidem fisierul
18 fileEditText.Close();
19 }
20
21 CDialog::OnOK();
22 }

```

- ❶ Întregul conținut al controlului de editare poate fi salvat printr-un singur apel al funcției `Write()`, deoarece lungimea sa este acum cunoscută.

În listingul 23.7, apelul `UpdateData()` din linia 6 înscrie în variabila membru `m_EditBox` conținutul curent al controlului de editare. Obiectul fișier declarat în linia 9 este deschis în liniile 10 și 11, specificându-se indicatorii `CFile::modeCreate` și `CFile::modeWrite` pentru crearea și suprascrierea fișierului existent (sau crearea unui nou, dacă nu există). Funcția `Write()` apelată în linia 14 salvează întregul conținut al șirului `m_EditBox`, convertit în prealabil la un buffer (`LPCTSTR`). Este mult mai ușor decât la citire, deoarece întregul șir poate fi scris dintr-o dată pentru că știți deja exact câți octeți trebuie scriși. Fișierul este închis de apelul funcției `Close()` din linia 18, încheindu-se operația de scriere.

#### VEDEȚI...

- Pentru mai multe informații despre controalele de editare și maparea șirurilor, vedeți pagina 100.
- Pentru amănunte despre proiectarea casetelor de dialog, vedeți pagina 94.
- Pentru detalii privind transferul datelor între controalele casetei de dialog și variabilele membru, vedeți pagina 207.

## Manipularea poziției curente în fișier

Așa cum am precizat mai devreme, există o poziție curentă în cadrul fișierului care este considerată poziție de început pentru funcțiile `Read()` și `Write()`. Această poziție poate fi manipulată, procedeu folosit frecvent la scrierea în diferite locuri dintr-un fișier a unor informații de tip înregistrare, cu dimensiune constantă. Poziția curentă poate fi determinată apelând funcția `GetPosition()` a unui obiect fișier, valoarea `DWORD` întoarsă reprezentând poziția actuală a cursorului asociat fișierului.

Această poziție poate fi totodată modificată prin intermediul funcțiilor de poziționare, `Seek()`, `SeekToBegin()` și `SeekToEnd()`, care plasează cursorul într-o poziție anume, la începutul sau, respectiv, la sfârșitul fișierului. Singura dintre aceste funcții care necesită parametri este funcția `Seek()`, aceasta primind doi parametri. Primul este o valoare `loff` de tip `LONG`, reprezentând un deplasament exprimat ca număr de octeți. Al doilea parametru poate fi oricare din indicatorii enumerați în tabelul 23.2, indicând că poziționarea se face relativ la începutul fișierului, la sfârșitul lui sau la poziția curentă.

**TABELUL 23.2 Indicatorii posibili pentru funcția `CFile::Seek()`**

| Indicator                   | Tipul de poziționare                                                          |
|-----------------------------|-------------------------------------------------------------------------------|
| <code>CFile::begin</code>   | Noua poziție este la <code>loff</code> octeți de la începutul fișierului      |
| <code>CFile::end</code>     | Noua poziție este la <code>loff</code> octeți înainte de sfârșitul fișierului |
| <code>CFile::current</code> | Noua poziție este la <code>loff</code> octeți de la poziția curentă           |

Puteți transmite valori negative funcției `Seek` atunci când vreți să vă poziționați relativ la sfârșitul fișierului sau la poziția curentă. De exemplu, pentru a vă deplasa înapoi cu 50 de octeți față de poziția curentă în fișier, ați adăuga linia următoare:

```
LONG loffset = fileMyFile.Seek(-50, CFile::current);
```

#### Cauze `CFileException` ale eșecului funcției `Seek()`

Dacă funcția `Seek()` eșuează dintr-un motiv oarecare, este generat un obiect `CFileException`. Codul corespunzător cauzei eșecului (aflat în `m_cause`) va fi în mod normal `CFileException::badSeek`, indicând că poziția solicitată nu este adecvată pentru fișier. Este posibil, de asemenea, să obțineți un cod `CFileException::invalidFile` la încercarea de efectuare a unei operații `Seek()` pentru un fișier care nu a fost deschis.

De asemenea, în cazul în care poziționarea reușește, `Seek` întoarce noua poziție curentă relativ la începutul fișierului. Dacă funcția `Seek()` eșuează (de regulă pentru că ați încercat să vă poziționați înainte de începutul sau după sfârșitul fișierului), este generată o excepție.

## Obținerea de informații despre fișiere

Există o serie de funcții care furnizează informații despre fișierul deschis curent. `GetLength()` întoarce o valoare `DWORD` care reprezintă lungimea curentă a întregului fișier (o funcție înrudită, `SetLength()`, trunchiază sau extinde fișierul la dimensiunea specificată ca parametru). `GetStatus()` întoarce informații privind momentul creării și al ultimei modificări, precum și atribute de citire și scriere. Funcția are două forme. Una dintre acestea operează asupra unui fișier deja deschis, înscriind informațiile într-un obiect `CFileStatus` transmis prin referință ca parametru. Cealaltă primește ca prim parametru numele fișierului și plasează informațiile obținute despre fișierul specificat în obiectul `CFileStatus` transmis prin referință ca al doilea parametru. De exemplu, dacă doriți să aflați anumite informații despre fișierul `C:\MyFile.txt`, cum ar fi momentul ultimei modificări, ați putea să adăugați liniile de mai jos după secvența de salvare a fișierului din funcția de tratare `OnOK()`:

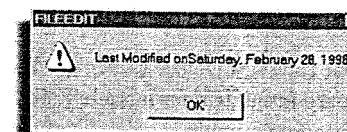
```
CFileStatus statusEditText;
CFile::GetStatus("C:\\MyFile.txt", statusEditText);
AfxMessageBox(_T("Last Modified on")+
statusEditText.m_mtime.Format("%A, %B %d, %Y"));
```

Apelul `GetStatus()` este precedat de operatorul de domeniu `CFile::` din cauză că această funcție este statică și nu are nevoie de un obiect atunci când îi este transmis numele de fișier. Dacă ați avea un fișier deschis, ați folosi în loc versiunea `fileMyOpenFile.GetStatus(statusEditText)`. Informațiile obținute sunt înscrise în obiectul `statusEditText`. Unul dintre membrii săi, `m_mtime`, conține ora și data ultimei modificări, fiind formatat prin intermediul funcției `CTime::Format()` în sensul afișării unui șir de forma `Last Modified on Saturday, February 28, 1998`.

Dacă asamblați și rulați aplicația după adăugarea liniilor de mai sus, odată ce efectuați clic pe butonul OK veți vedea caseta de mesaj înfățișată în figura 23.3.

FIGURA 23.3

Afișarea datei ultimei modificări a fișierului în exemplul de editor de text.



Ceilalți membri ai clasei `CFileStatus` pe care îi puteți folosi sunt prezentați în tabelul 23.3.

**TABELUL 23.3 Membri ai clasei `CFileStatus` corespunzători atributelor unui fișier**

| Atributul și tipul             | Descriere                                                          |
|--------------------------------|--------------------------------------------------------------------|
| <code>CTime m_mtime</code>     | Data și ora ultimei modificări                                     |
| <code>CTime m_atime</code>     | Data și ora ultimei accesări                                       |
| <code>CTime m_ctime</code>     | Data și ora creării                                                |
| <code>LONG m_size</code>       | Dimensiunea în octeți a fișierului                                 |
| <code>BYTE m_attribute</code>  | Atribute de citire, scriere și alte atribute (vedeți tabelul 23.4) |
| <code>char m_szFullName</code> | Numele fișierului și calea completă                                |

Majoritatea membrilor au o semnificație evidentă, dar `m_attribute` conține mai mulți indicatori care pot fi testați cu ajutorul operatorului `ȘI` logic, `&`. Acești indicatori sunt prezentați în tabelul 23.4 și pot fi testați astfel:

```
if (statusEditText.m_attribute & CFile::readOnly)
 AfxMessageBox("File is Read Only!");
```

#### Limitări impuse de Windows asupra numelor și căilor de fișiere

Multe dintre structurile Windows definesc numele și calea unui fișier ca un vector de caractere cu `MAX_PATH` elemente. În prezent, `MAX_PATH` este definit la valoarea 260, ceea ce înseamnă că în Windows numele și calea unui fișier nu trebuie să depășească 260 de caractere.

Secvența de mai sus verifică dacă fișierul conține atributul de protejare la scriere; toți ceilalți indicatori sunt ignorați, deoarece operatorul `ȘI` logic anulează toți biții, mai puțin bitul `CFile::readOnly` (valoarea 1). Dacă acest indicator este prezent, rezultatul va fi nenul, deci `TRUE`; altfel, rezultatul este zero, sau `FALSE`.

**TABELUL 23.4 Indicatori pentru membrul `m_attribute` al clasei `CFileStatus`**

| Indicator              | Valoare hexa      | Descriere                                            |
|------------------------|-------------------|------------------------------------------------------|
| <code>normal</code>    | <code>0x00</code> | Fișier obișnuit, fără atribute speciale              |
| <code>readOnly</code>  | <code>0x01</code> | Este permisă doar citirea din fișier, nu și scrierea |
| <code>hidden</code>    | <code>0x02</code> | Fișierul este ascuns                                 |
| <code>system</code>    | <code>0x04</code> | Fișierul este un fișier de sistem                    |
| <code>volume</code>    | <code>0x08</code> | Fișierul este un volum de disc                       |
| <code>directory</code> | <code>0x10</code> | Fișierul este un director de pe disc                 |
| <code>archive</code>   | <code>0x20</code> | Fișierul este o arhivă                               |

Există și alte funcții care furnizează informații despre fișiere, așa cum este `GetFileName()`, care întoarce doar numele fișierului (nu și calea de directoare). `GetFilePath()` întoarce numele și calea fișierului curent. `GetFileTitle()` întoarce

numele fișierului, fără cale sau extensie, deci `GetFileTitle()` aplicată fișierului `C:\MyFile.txt` va întoarce `MyFile`.

## Redenumirea și ștergerea fișierelor

Există două funcții statice care au ca efect redenumirea și ștergerea fișierelor: `Rename()` și `Remove()`. `Rename()` primește doi parametri: vechiul nume al fișierului și noul său nume. Pentru că este o funcție statică, nu este nevoie să deschideți vreun fișier, fiind suficient să prefixați apelul cu operatorul de domeniu:

```
CFile::Rename("C:\\MyFile.txt", "C:\\NewName.txt");
```

#### Funcțiile statice din C++

O funcție statică este o funcție membru a unei clase C++ care nu are nevoie de un obiect context. În timp ce majoritatea funcțiilor membru din C++ necesită un pointer `this`, care indică obiectul în cauză, pentru a putea referi variabilele locale, o funcție statică nu utilizează nici una dintre variabilele membru ale obiectului și poate fi apelată în lipsa oricărei instanțe a obiectului.

Funcția `Remove()` este de asemenea statică și primește numele fișierului care trebuie șters, ca aici:

```
CFile::Remove("C:\\MyFile.txt");
```

Ambele funcții generează o excepție în cazul în care eșuează dintr-un motiv oarecare.

## Alte clase derivate din `CFile`

Clasa `CFile` pune la dispoziție funcționalitatea de bază legată de manipularea fișierelor. Pentru o funcționalitate mai specializată sunt disponibile o serie de clase derivate. Nu avem spațiu suficient pentru o prezentare pe larg, așa că mă voi rezuma la a menționa câteva dintre aceste clase și scopul lor.

- `CMemFile` oferă un fișier de memorie. Acesta poate fi util atunci când aveți nevoie de funcționalitatea specifică accesului la fișiere, dar doriți viteza oferită de lucrul cu memoria, prin contrast cu viteza hard-diskului.
- `CSharedFile` este un caz particular de `CMemFile` care alocă și se atașează la o memorie partajată. Acesta este folosit de regulă în comunicațiile inter-procese, așa cum este transferul înspre și dinspre Clipboard. Pentru un exemplu concret, vedeți secțiunea „Transferul de date prin Clipboard”, prezentată mai târziu în acest capitol.
- `COleStream` este clasa de bază pentru operații cu fluxuri OLE. De aici sunt derivate mai multe clase pentru legare, încapsulare și automatizare OLE, printre care `CMonikerFile`, `CAsyncMonikerFile`, `CDataPathProperty` și `CCachedDataPathProperty`.
- `CSocketFile` vă permite să tratați conexiunile de rețea asemeni fișierelor, ajutând la transferul prin rețea al datelor documentelor prin intermediul serializării (atunci când se folosește împreună cu un obiect `CArchive`).

- `CStdioFile` încapsulează vechile concepte de `stdin` și `stdout` din C, permițând redirectarea liniei de comandă. Folosește indicatorul de deschidere în mod text pentru a efectua conversia între `<CR>` și `<CR><LF>` (caracterele retur de car și avans de linie). Această clasă este specializată mai departe de `CInternetFile`, care aduce funcționalitatea de bază pentru `CHttpFile`, destinată documentelor în stil *HTML World Wide Web*, și pentru `CGopherFile`, destinată formatului *server Gopher*, aproape dispărut din Internet.
- Fără a fi derivată, `CRecentFileList` este asociată cu clasa `CFile` și oferă funcționalitatea privind lista fișierelor cel mai recent utilizate, despre care am vorbit în acest capitol, în secțiunea legată de serializare. Această clasă poate fi folosită direct pentru implementarea propriilor liste și meniuri cu fișierele utilizate recent.

#### VEDEȚI...

► Pentru mai multe informații privind clasele pentru suportul Internet, vedeți pagina 454.

## Transferul de date prin Clipboard

Decuparea și lipirea datelor dintr-o aplicație în alta a devenit unul dintre conceptele consacrate ale paradigmei Windows de interfață standard cu utilizatorul. Prin înregistrarea tipurilor de date cu care operează, o aplicație poate să ofere și să accepte date dintr-o altă instanță a sa sau dintr-o aplicație complet diferită, dar care lucrează cu un format cunoscut.

#### Noile și vechile metode de transfer prin Clipboard

Odată cu apariția tehnicilor OLE, vechile mecanisme DDE (Dynamic Data Exchange – schimb dinamic de date) pentru transfer prin Clipboard au fost înlocuite de sursele de date și obiectele de date OLE. Este mai bine să folosiți metodele OLE în locul DDE, deoarece acestea se bucură de un suport mai larg, sunt mai flexibile și permit implementări mai facile ale facilităților de tragere și plasare.

## Stabilirea formatelor de date pentru Clipboard

Primul lucru care trebuie făcut înainte de a putea să transferați date înspre sau dinspre Clipboard este să determinați aplicația să înregistreze formatele de date cu care poate să opereze. Acestea pot fi formate standard, dintre care o parte sunt prezentate în tabelul 23.5, sau pot fi formate proprii, speciale, care se pot înregistra cu ajutorul funcției `RegisterClipboardFormat()`. În ambele situații, formatul este o valoare `UINT`.

**TABELUL 23.5 Formate standard pentru Clipboard**

| Indicator de format    | Descrierea formatului                                                                                                                                                                                                |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>CF_TEXT</code>   | Probabil cel mai uzual format, folosit la transferul textului, sfârșitul fiecărei linii fiind marcat prin secvența de caractere <code>&lt;CR&gt;</code> și <code>&lt;LF&gt;</code> (retur de car și avans de linie). |
| <code>CF_BITMAP</code> | Datele reprezintă un indicator asociat unui bitmap.                                                                                                                                                                  |
| <code>CF_DIB</code>    | Datele reprezintă o imagine stocată ca bitmap independent de dispozitiv.                                                                                                                                             |

| Indicator de format          | Descrierea formatului                                                                                  |
|------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>CF_TIFF</code>         | Datele reprezintă o imagine în formatul TIFF.                                                          |
| <code>CF_WAVE</code>         | Datele sunt în format de semnal audio cu modulare standard prin impulsuri (PCM).                       |
| <code>CF_PALETTE</code>      | Datele reprezintă un indicator asociat unei palete de culori.                                          |
| <code>CF_METAFILEPICT</code> | Datele reprezintă o imagine în format meta-fișier, definită de o structură <code>METAFILEPICT</code> . |

Ați putea să adăugați la exemplul `Persist`, prezentat mai devreme în acest capitol, facilități de decupare și lipire prin intermediul unui format blob propriu definit. Puteți să refolosiți codul implementat pentru serializare în scopul efectuării transferurilor prin Clipboard. Pentru că obiectele blob nu reprezintă un format suportat de către Clipboard (încă!), trebuie să înregistrați acest format printr-un apel `RegisterClipboardFormat()`.

`RegisterClipboardFormat()` primește un singur parametru, un șir care identifică unic formatul de date, întorcând o valoare `UINT` care reprezintă formatul înregistrat. Programul care înregistrează prima dată șirul de caractere va obține o valoare unică; înregistrări ulterioare ale aceluiași format vor întoarce această valoare. Se asigură, astfel, că instanțe multiple ale aceleiași aplicații vor opera cu un același format de Clipboard și vor putea să determine dacă suportă un format specificat.

#### Formate de Clipboard

Un format de Clipboard este un tip de date prestabilit care permite programelor ce înțeleg acest tip de date să determine dacă în Clipboard există astfel de date. Există o funcție `CountClipboardFormat()` care întoarce un întreg reprezentând numărul de formate distincte înregistrate curent în Clipboard. O altă funcție, `EnumClipboardFormats()`, întoarce pe rând fiecare format. Pentru început trebuie să transmiteți zero acestei funcții, obținând astfel primul format. Transmițând apoi acest format într-un apel ulterior `EnumClipboardFormat()` veți obține următorul format.

Apelul `RegisterClipboardFormat()` poate fi efectuat în cadrul constructorului clasei `CPersistDoc`, astfel încât formatul să fie înregistrat imediat ce se creează documentul.

```
CPersistDoc::CPersistDoc()
{
 // TODO: add one-time construction code here
 m_uClipFormat =
 RegisterClipboardFormat("PersistClipFormat");
}
```

Observați că formatul întors este reținut într-o variabilă membru a obiectului document în scopul unei utilizări ulterioare. Această variabilă membru poate fi inserată în definiția clasei, după containerul `m_BlobArray` de obiecte blob:

```
CObArray m_BlobArray;
UINT m_uClipFormat; // ** Noul membru de format
```

Acesta este tot codul necesar pentru înregistrarea în Clipboard a propriului tip `PersistClipFormat`. Dacă ați fi adăugat suport pentru un tip standard, așa cum este `CF_TEXT`, nu ar mai fi fost necesară nici o înregistrare; `CF_TEXT` ar fi identificatorul de format pe care l-ați transmite funcțiilor de transfer prin Clipboard.

## Copierea datelor în Clipboard

Următorul lucru pe care trebuie să-l implementați este partea de decupare și copiere a transferului prin Clipboard. Decuparea nu este decât un caz particular de copiere, în care datele documentului sunt șterse la final. Codul pentru copiere poate fi implementat într-o funcție de tratare corespunzătoare opțiunii **Copy** din meniul **Edit**. Folosiți **ClassWizard** pentru a adăuga în clasa document o funcție de tratare, `OnEditCopy()`, pentru această opțiune de meniu și o altă funcție de tratare, `OnEditCut()`, pentru opțiunea **Cut**. Dacă aveți nevoie de asistență în acest scop, instrucțiunile necesare pot fi găsite în secvența de pași „Adăugarea unei funcții de tratare pentru o comandă de meniu cu **ClassWizard**”, în capitolul 13, „Lucrul cu meniuri”.

### Transferul prin Clipboard sub forma schimbului dinamic de date

Înainte de apariția OLE, schimbul dinamic de date (DDE) era metoda fundamentală de transfer al datelor prin Clipboard (fiind utilizate segmente de memorie partajate). Funcțiile DDE există încă, oferind compatibilitatea înapoi, dar este de preferat să folosiți OLE deoarece aduce mai multă flexibilitate și înlesnește și standardizează operațiile de trageră și plasare între aplicații diferite.

Odată create funcțiile de tratare, puteți să adăugați codul care efectuează copierea în Clipboard, cod prezentat în listingul 23.8. Procesul implementat are trei etape fundamentale: crearea unui obiect sursă de date OLE, serializarea datelor documentului într-un fișier de memorie partajat și prezentarea datelor către Clipboard.

### LISTINGUL 23.8 LST24\_8.CPP – Implementarea opțiunii de copiere prin serializarea datelor documentului înspre Clipboard

```
1 void CPersistDoc::OnEditCopy()
2 {
3 // Cream un obiect sursa de date
4 COleDataSource* pDataSource = new COleDataSource;
5
6 // Cream un fisier partajat si-i atasez un obiect CArchive
7 CSharedFile fileClipCopy;
8 CArchive arClipCopy(&fileClipCopy, CArchive::store);
9
10 // Serializam documentul in aceasta arhiva
11 Serialize(arClipCopy);
12 arClipCopy.Close();
13
14 // Obtinem indicatorul global de memorie pentru fisier
15 HANDLE hGlobalMem = fileClipCopy.Detach();
```

```
16 if (hGlobalMem)
17 {
18 // Preluam datele la nivel global, cu formatul unic
19 pDataSource->CacheGlobalData(1
20 m_uClipFormat, hGlobalMem);
21 // Prezintam datele catre clipboard
22 pDataSource->SetClipboard();
23 }
24 else AfxMessageBox("Can't alloc memory for copy");
25 } 2
```

1 `CacheGlobalData()` este una din multele soluții pentru stabilirea unei referințe la date în cadrul obiectului `CDataSource`.

2 Nu este necesar să ștergeți obiectul sursă de date alocat, pentru că Clipboard-ul o va face automat.

Primul lucru pe care trebuie să-l faceți este să creați un obiect `COleDataSource`, ca în linia 4. Acest obiect OLE are rolul de a oferi diferite tipuri de date în scopul transferului sub mai multe forme diferite – nu numai transferul prin Clipboard, dar și prin trageră și plasare, precum și multe alte tipuri de transfer de date.

Obiectul `CSharedFile`, declarat în linia 7, este un tip special de fișier de memorie. Constructorul implicit deschide automat un fișier de memorie și alocă o zonă de memorie partajată globală în care va fi stocat conținutul fișierului.

Linii 8 și 11 utilizează în mod inteligent serializarea pe care ați implementat-o deja în clasa `CBlob` și în cadrul documentului pentru a stoca datele în fișierul de memorie partajat. În acest scop, în linia 8 se creează un obiect local `CArchive` care se atașează la fișierul de memorie partajat; este specificat indicatorul `CArchive::store`, ceea ce va face ca funcția `IsStoring()` a arhivei să întoarcă `TRUE`, marcând faptul că datele sunt înscrise în arhivă.

Arhiva este închisă în linia 12 pentru a determina transferul conținutului său în cadrul fișierului, astfel că memoria globală va conține întreg documentul. Dacă este necesară mai multă memorie partajată, `CShareFile` alocă automat memoria necesară.

Linii 15 și 16 detașează fișierul de la memoria partajată, întorcând un indicator care, dacă este valid, va fi utilizat în cadrul sursei de date.

În linia 19, în apelul funcției `CacheGlobalData()` a obiectului `CDataSource` se transmite indicatorul global de memorie și un identificator unic de format. Sursa de date este gata pentru a fi prezentată către Clipboard, ceea ce se întâmplă în linia 22, prin apelul membrului `SetClipboard()` al obiectului sursă de date.

Este necesară includerea câtorva fișiere adiționale în care sunt definite obiectele OLE și funcțiile pentru Clipboard. Directivele corespunzătoare pot fi plasate imediat înainte de definiția clasei document din fișierul de antet `PersistDoc.h`, ca mai jos:

```
#include <afxadv.h> // ** Nou fișier de antet necesar
#include <afxole.h> // ** Nou fișier de antet necesar
```



```
class CPersistDoc : public CDocument
{ ...
```

Pentru că acum utilizați OLE, va trebui să inițializați bibliotecile OLE și COM. Aceasta se face ușor, adăugând apelul `AfxOleInit()` de mai jos după apelul `AfxEnableControlContainer()` din funcția membru `InitInstance()` a clasei `CPersistApp`:

```
BOOL CPersistApp::InitInstance()
{
 AfxEnableControlContainer();
 AfxOleInit(); // ** Noul apel pentru initializarea bibliotecii OLE
```

Cu aceasta am încheiat implementarea copierii în Clipboard. Toate acestea pot fi refolosite la implementarea operației de decupare, apelând funcția de tratare a opțiunii **Copy** din cadrul funcției de tratare a opțiunii **Cut**, urmând ca apoi să ștergeți toate datele documentului. Așadar, funcția de tratare `OnEditCut()` ar putea fi următoarea:

```
void CPersistDoc::OnEditCut()
{
 OnEditCopy();
 DeleteBlobs();
 UpdateAllViews(NULL);
}
```

#### Codul întors de `AfxOleInit()`

Pentru simplitate, codul întors de `AfxOleInit()` nu este testat aici. Dar dacă inițializarea OLE eșuează, `AfxOleInit()` întoarce valoarea zero; aceasta arată, de obicei, că versiunile bibliotecilor DLL instalate în sistem sunt necorespunzătoare. Pentru că aproape toate celelalte aplicații vor fi eșuate deja într-un astfel de caz, de multe ori se presupune că `AfxOleInit()` a reușit.

## Lipirea datelor din Clipboard

Cealaltă parte a transferului prin Clipboard este implementarea operației de lipire. Aceasta folosește un obiect OLE corespunzător, având tipul `COleDataObject`. Acest obiect poate să verifice dacă în Clipboard există date în format adecvat, preluându-le în caz că da. Din acest punct, totul se reduce la serializarea inversă pentru aducerea obiectelor blob din Clipboard în cadrul documentului, prin intermediul unei alte combinații de obiecte `CArchive` și `CSharedFile`, așa cum o ilustrează listingul 23.9.

#### LISTINGUL 23.9 LST24\_9.CPP – Lipirea datelor din Clipboard în cadrul documentului prin serializarea datelor oferite de `CDataObject`

```
1 void CPersistDoc::OnEditPaste()
2 {
3 // Cream un obiect de date si-l atasam la clipboard
4 COleDataObject oleTarget;
5 oleTarget.AttachClipboard();
```

```
6 if (oleTarget.IsDataAvailable(m_uClipFormat))
7 {
8 // Obținem un indicator global de memorie
9 HANDLE hGlobalMem =
10 oleTarget.GetGlobalData(m_uClipFormat);
11 if (hGlobalMem)
12 {
13 // Stergem datele din document
14 CSharedFile fileTarget;
15 fileTarget.SetHandle(hGlobalMem);
16 CArchive arTarget(&fileTarget, CArchive::load);
17 DeleteBlobs();
18
19 // Efectuam serializarea si actualizam afisarea
20 Serialize(arTarget);
21 UpdateAllViews(NULL);
22 }
23 }
24 }
```

❶ Operația de lipire este practic inversul operației de copiere, datele fiind încărcate înapoi din memoria partajată.

În acest listing, obiectul `oleTarget` de tip `COleDataObject` este declarat în linia 4 și atașat la Clipboard în linia 5. Dacă funcția `IsDataAvailable()` a obiectului de date întoarce valoarea `TRUE` atunci când verifică prezența unor date având formatul `m_uClipFormat` (în linia 6), înseamnă că aceste date există în Clipboard și pot fi transferate.

Indicatorul global la memoria partajată este extras din cadrul obiectului `COleDataObject` în linia 9, iar în linia 14 este declarat un fișier de memorie partajat, `fileTarget`, care este asociat indicatorului determinat. În linia 16 este creat un obiect `arTarget`, de tip `CArchive`, fiind pregătit pentru încărcare. Obiectele blob existente curent sunt șterse în linia 17, iar noile obiecte sunt încărcate din fișierul de memorie în linia 20. În final, linia 21 determină actualizarea afișării pe baza noilor date transferate.

Puteți să adăugați cu ușurință o funcție de tratare a interfeței cu utilizatorul (vedeți secțiunea „Activarea și dezactivarea opțiunilor de meniu” din capitolul 13, „Lucrul cu meniuri”) pentru opțiunea de meniu **Paste**, astfel încât aceasta să fie activată doar dacă în Clipboard există date corespunzătoare:

```
void CPersistDoc::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
 COleDataObject oleTarget;
 oleTarget.AttachClipboard();
 pCmdUI->Enable(oleTarget.IsDataAvailable(m_uClipFormat));
}
```

Funcția verifică dacă în Clipboard există date corespunzătoare, activând elementul de meniu în cazul în care apelul `IsDataAvailable()` întoarce valoarea `TRUE`; altfel, funcției



`Enable()` a obiectului `pCmdUI` îi este transmisă valoarea `FALSE`, determinând afișarea voalată a opțiunii de meniu.

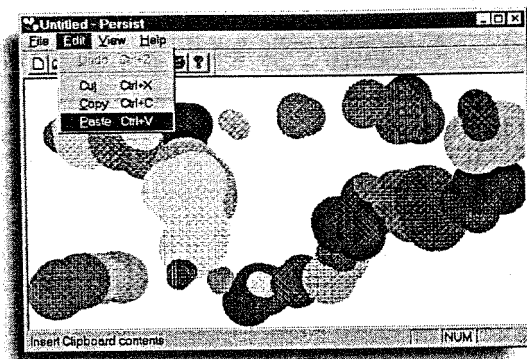
#### Funcția `AttachClipboard()`

`AttachClipboard()` conectează un obiect `COleDataObject` la datele existente în Clipboard. Datele vor fi blocate în Clipboard până la apelul funcției `Release()` a obiectului `COleDataObject`. Această funcție este apelată automat de către destructorul clasei `COleDataObject` în momentul în care domeniul de existență al obiectului se încheie, la sfârșitul funcției.

Cu aceasta am implementat întreg codul necesar pentru operațiile de decupare și lipire. Puteți să asamblați și să rulați aplicația, iar pentru a o testa lansați două instanțe, creați câteva obiecte blob, copiați-le sau decupați-le dintr-o instanță și lipiți-le în cealaltă instanță, așa cum se vede în figura 23.4. Serializarea în fișier funcționează în continuare, așa că puteți să încărcați un fișier blob existent sau să salvați un fișier după o operație de lipire.

FIGURA 23.4

Decuparea și lipirea obiectelor blob între două instanțe ale programului `Persist`.



#### VEDEȚI...

- Pentru amănunte privind activarea și dezactivarea opțiunilor de meniu, vedeți pagina 277.
- Pentru mai multe informații despre suportul pentru OLE și COM, vedeți pagina 589.

## CAPITOLUL

# 24

## Utilizarea bazelor de date și a reprezentărilor de tip înregistrare

Configurarea unei surse de date ODBC

Accesarea bazelor de date în cadrul aplicațiilor MFC

Interogarea mulțimilor de înregistrări și accesarea rezultatelor

Adăugarea, ștergerea și editarea înregistrărilor dintr-o bază de date

Dezvoltarea unui proiect cu o reprezentare de tip înregistrare

## Utilizarea bazelor de date

Organizațiile cu profil economic acumulează o cantitate mare de informații în urma activităților comerciale de zi cu zi. Sunt înregistrate detalii despre clienți și furnizori, despre vânzări și necesarul de stocuri și așa mai departe. Marea majoritate a corporațiilor optează pentru stocarea acestor date vitale în cadrul bazelor de date sau, mai exact, într-un *DMS (Data Management System - sistem de gestionare a bazelor de date)*. Un sistem de gestionare a bazelor de date este un produs software care stochează datele într-o structură bine organizată și oferă mijloace eficiente de accesare și actualizare a acestor date.

Există două tipuri diferite de baze de date: baze de date relaționale și baze de date obiectuale. Diferența dintre ele provine din modul conceptual în care sunt gestionate datele. Bazele de date relaționale au la bază tipuri de date simple, recunoscute (caractere, șiruri, întregi, etc.) și nu permit crearea unor noi tipuri de date. Bazele de date obiectuale operează cu tipurile de date la un nivel mai înalt, permițând crearea prin definire a unor noi tipuri de date. Aceste obiecte corespund celor create într-un limbaj de programare orientat pe obiect; spre exemplu, un obiect stocat într-o bază de date obiectuală ar putea fi o persoană sau un automobil. Capitolul de față se oprește asupra utilizării bazelor de date relaționale, deoarece acestea sunt folosite de mai multă vreme și pe o scară mai largă.

## Utilizarea bazelor de date relaționale

Înainte de apariția sistemelor de gestionare a bazelor de date, aplicațiile scrise foloseau structuri de fișiere proprietare. Numai cei care proiectau și programau sistemul știau exact cum erau dispuse datele. Aceasta înseamnă că atunci când utilizatorii sistemului aveau nevoie să manipuleze datele într-o manieră aparte, trebuiau de regulă să solicite o extindere a sistemului, ceea ce de multe ori necesita timp.

Bazele de date relaționale au eliminat în bună parte această problemă printr-o standardizare a modului de stocare a datelor. Aceste date sunt organizate pe tabele, coloane și linii. De exemplu, o bază de date ar putea să conțină o tabelă Clienți cu coloanele Numele Firmei și Număr de Telefon. Liniile reprezintă elementele propriu-zise ale tabelului, corespunzătoare entităților stocate. De pildă, tabela Clienți poate avea o linie pentru Teora și o alta pentru Microsoft. Structura acestor tabele și coloane este reținută în *schema bazei de date*. Această schemă definește tipul de date al fiecărei coloane și legăturile dintre tabele.

## Utilizarea conectivității deschise a bazelor de date

Există mai mulți producători care oferă baze de date relaționale. Fiecare bază de date are propria sa structură și, în consecință, un set propriu de funcții API. Aceasta poate presupune o curbă de învățare abruptă în încercarea de a înțelege funcționarea bazei de date a unui anumit producător. Din ce în ce mai mult, se cere ca aplicațiile să poată lucra cu orice bază de date. Pentru a putea dezvolta aplicații care să utilizeze o bază de date relațională, indiferent de producătorul acesteia, este nevoie de o metodă generică de programare. O astfel de metodă există și se numește ODBC (Open Database Connectivity – conectivitate deschisă a bazelor de date).

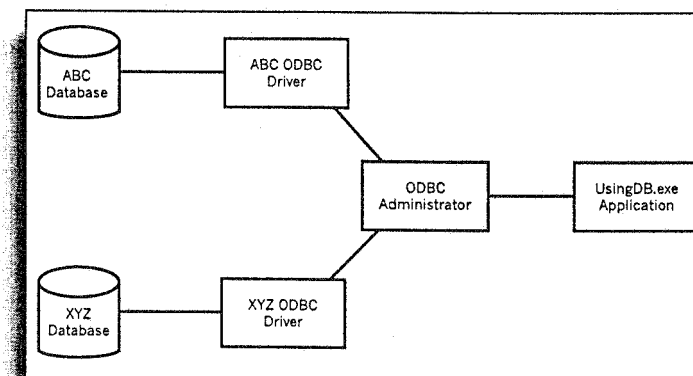
### Distribuirea aplicațiilor ODBC

În cazul în care dezvoltați aplicații de baze de date care folosesc ODBC, există o serie de fișiere adiționale care trebuie livrate împreună cu fișierul executabil. Aceste fișiere adiționale se află în directorul OS\System de pe CD-ul Visual C++. Consultați fișierul REDISTRB.WRI aflat în același director pentru a afla mai multe informații.

ODBC aduce o interfață de programare standard prin care pot fi utilizate diferite tipuri de baze de date relaționale. În acest scop este necesară existența unui intermediar care să traducă apelurile ODBC standard în apeluri de funcții specifice bazei de date. Acest intermediar este driverul ODBC, oferit de către producătorul bazei de date sau de către o firmă terță specializată în astfel de produse. ODBC a fost adoptat ca standard și driverul ODBC există acum pentru toate bazele de date răspândite. Dumneavoastră va trebui să instalați driverul ODBC corespunzător bazei de date pe care o utilizați. Procedura de instalare a unor aplicații Microsoft, precum Office și Visual Studio, vă permite instalarea celor mai folosite drivere ODBC produse de Microsoft; pentru a utiliza o altă bază de date, contactați producătorul acesteia în legătura cu disponibilitatea unui driver ODBC. Figura 24.1 ilustrează modul în care se realizează legătura dintre o bază de date și aplicația dumneavoastră, prin intermediul driverului ODBC.

FIGURA 24.1

Accesarea bazelor de date prin intermediul ODBC.



Comenzile sunt transmise driverului ODBC și pasate mai departe bazei de date cu ajutorul SQL (Structured Query Language – limbaj structurat de interogare). Acest limbaj a fost dezvoltat anume pentru accesarea bazelor de date și este acum standardul *de facto*. Există aproape tot atâtea variante de SQL câte baze de date sunt, fiecare producător aducând propriile îmbunătățiri. Dar există cerințe minime care asigură un set de comenzi suportat de orice driver ODBC. Aceasta nu împiedică driverele să suporte comenzi care nu sunt standard, deoarece ODBC oferă o metodă de scurt-circuitare ce permite execuția directă a instrucțiunilor SQL. Cu toate acestea, utilizarea unor astfel de comenzi în afara standardului înseamnă că aplicația dumneavoastră își pierde capacitatea de a accesa bazele de date ale altor producători, tocmai această capacitate fiind avantajul major adus de ODBC.

**Limbajul structurat de interogare**

Limbajul structurat de interogare (SQL) este un limbaj textual folosit de către bazele de date relaționale ca standard pentru extragerea, actualizarea și gestionarea datelor.

SQL conține trei tipuri de comenzi: DDL, DML și DCL. Comenzile DDL (Data Definition Language – limbaj pentru definirea datelor) sunt folosite pentru crearea și modificarea schemelor de baze de date. Exemple de comenzi DDL sunt **CREATE DATABASE** și **CREATE TABLE**. Comenzile DML (Data Manipulation Language – limbaj pentru manipularea datelor) sunt folosite pentru interogarea și manipularea informațiilor. Există patru comenzi DML: **SELECT**, **INSERT**, **UPDATE** și **DELETE**. Fiecare dintre acestea primește parametri care specifică tabela(-ele) și coloana(-ele) implicate. Comenzile DCL (Data Control Language – limbaj pentru controlul datelor) sunt folosite pentru acordarea unor drepturi de acces specifice diferiților utilizatori.

**Configurarea unei surse de date**

Prima sarcină legată de utilizarea ODBC este configurarea unei surse de date. O sursă de date indică programului unde se află fișierele care conțin baza de date și ce driver ODBC trebuie folosit pentru interpretarea apelurilor API. Configurarea surselor de date se face cu ajutorul administratorului ODBC pentru sursele de date, accesibil din panoul de control.

**Utilizarea programului administrator ODBC adecvat**

S-ar putea ca în panoul de control să se afle două pictograme ODBC; în acest caz, folosiți-o pe cea a cărei imagine conține numărul 32, aceasta fiind destinată configurării accesului pe 32 de biți la bazele de date. Cealaltă pictogramă privește accesul pe 16 biți.

Pentru configurarea unei surse de date care va fi utilizată mai târziu, în exemplul de aplicație creat în acest capitol, urmați pașii aflați în secvența „Crearea și configurarea unei surse de date ODBC”. Această aplicație va utiliza fișierul `stdreg32.mdb`, care este o bază de date Acces oferită ca exemplu pe CD-ul Visual C++. La momentul editării acestei cărți, locația exactă pe CD a fișierului nu este cunoscută. Va trebui să găsiți acest fișier și să-l copiați de pe CD pe propriul hard-disc. Secvența de pași care urmează presupune că fișierul se află în directorul `C:\Databases\`.

**Crearea și configurarea unei surse de date ODBC**

1. În cadrul meniului **Start** accesibil de pe suprafața de lucru, deschideți submeniul **Settings** și apoi selectați **Control Panel**.
2. Efectuați un dublu clic (sau selectați și apăsați **Enter**) pe pictograma etichetată **32bit ODBC**. Va fi afișată caseta de dialog **ODBC Data Source Administrator**, înfățișată în figura 24.2.
3. Selectați pagina **User DSN** și efectuați un clic pe butonul **Add**. Este deschisă caseta de dialog **Create New Data Source**, ca în figura 24.3. Puteți să modificați opțiunile

asociate unei surse de date existente selectând numele său c  
**Sources** și efectuând un clic pe butonul **Configure**.

FIGURA 24.2

Caseta de dialog **ODBC Data Source Administrator**.

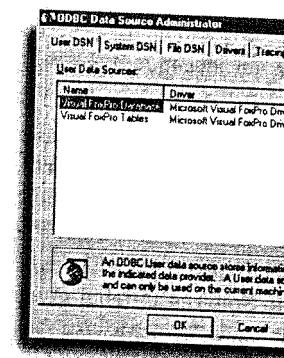
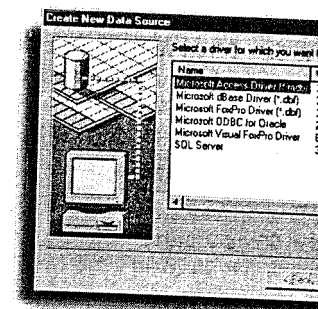


FIGURA 24.3

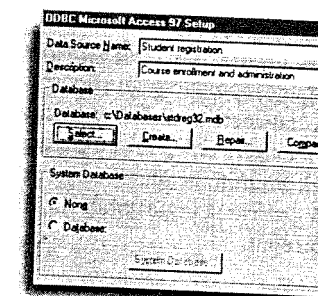
Caseta de dialog **Create New Data Source**.



4. Selectați numele driverului ODBC pentru care doriți să creați că acesta trebuie să corespundă tipului bazei de date. Pentru e **Microsoft Access Driver (\*.mdb)**. Efectuați un clic pe **Finish** dialog **ODBC Microsoft Access Setup**, ilustrată în figura 24.4

FIGURA 24.4

Caseta de dialog **ODBC Microsoft Access Setup**.



5. Introduceți un nume pentru sursa de date în cadrul casetei de editare **Data Source Name**. În cazul nostru, introduceți *Student Registration*. Puteți să specificați orice nume doriți, dar ar trebui să evitați existența a două surse de date având același nume.
6. Opțional, introduceți în caseta de editare **Description** o descriere pentru sursa de date. De exemplu, ați putea să introduceți *Course enrollment and administration*.
7. Efectuați un clic pe butonul **Select**; va fi afișată caseta de dialog **Select Database**.
8. Introduceți numele fișierului bază de date în caseta de editare **Database Name** sau selectați fișierul cu ajutorul listei de dosare **Directories**. Pentru exemplul nostru, selectați fișierul *stdreg32.mdb*. Efectuați un clic pe **OK**. Locația și numele bazei de date selectate sunt afișate în caseta de dialog **Setup**.
9. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ODBC Microsoft Access Setup**; în lista **User Data Sources** ar trebui să apară acum *Student Registration*.
10. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ODBC Data Source Administrator** și închideți fereastra panoului de control.

Observați că în caseta de dialog din figura 24.4 sunt prezente opțiuni de configurare ODBC specifice unei baze de date Microsoft Access. Alte baze de date vor avea propriile casete de dialog pentru configurare și ar putea necesita stabilirea unor opțiuni suplimentare.

#### Rezolvarea problemelor privind accesul la baza de date

Dacă aveți probleme cu accesul la baza de date, încercați să schimbați tipul sursei de date. Spre exemplu, dacă ați selectat **User DSN**, treceți la **System DSN**, și viceversa.

Remarcați, de asemenea, că puteți să configurați două tipuri de surse de date: **User DSN** (implicit) și **System DSN**. O sursă de date utilizator este accesibilă exclusiv utilizatorului care a creat-o, în timp ce o sursă de date sistem este la dispoziția oricărui utilizator care lucrează pe calculator.

## Generarea unei aplicații cu suport pentru bazele de date

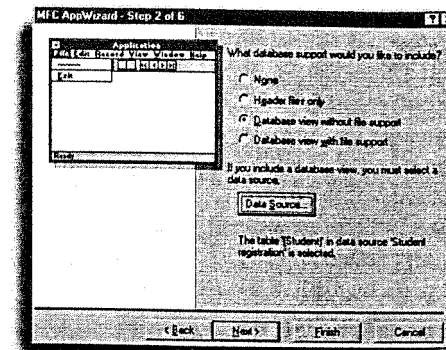
Modul în care se dezvoltă o aplicație care necesită accesarea unei baze de date depinde foarte mult de o serie de factori decisivi. Spre exemplu, trebuie aplicația să acceseze mai multe baze de date sau trebuie suportate și alte tipuri de fișiere? Toate aceste cazuri sunt posibile și trebuie luate în considerare, deoarece metoda cea mai simplă de integrare a suportului pentru baze de date este utilizarea opțiunilor oferite de AppWizard la momentul creării proiectului. Dacă vă amintiți, am menționat că opțiunile selectate în AppWizard nu mai pot fi schimbate după generarea proiectului, singurele soluții fiind editarea codului sau crearea proiectului de la început.

## Includerea suportului pentru bazele de date cu ajutorul AppWizard

Pasul 2 din AppWizard vă permite să alegeți una din patru opțiuni care oferă diferite nivele de suport pentru bazele de date (vedeți figura 24.5).

FIGURA 24.5

Opțiunile din AppWizard privind suportul pentru bazele de date.



Prima opțiune (**None**) este evidentă. Este exclus orice suport pentru bazele de date.

A doua opțiune (**Header Files Only**) are ca efect simpla adăugare în proiect a directivei `#include` pentru fișierul de antet care conține clasele `MFC CDatabase` și `CRecordset`. Acestea sunt clasele folosite pentru conectarea la o bază de date și accesarea informațiilor conținute.

#### Adăugarea într-un proiect existent a suportului pentru baze de date

Adăugarea de suport pentru baze de date în cadrul unui proiect care a fost creat inițial cu AppWizard și fără suport pentru baze de date se poate face prin simpla includere în `stdafx.h` a două fișiere de antet: `afxdb.h` și `afxdao.h`.

Opțiunile a treia (**Database View Without File Support**) și a patra (**Database View With File Support**) necesită ambele specificarea unei surse de date pentru proiect, având ca efect crearea automată a unor clase specifice, în funcție și de alte opțiuni. Diferența dintre ele este faptul că opțiunea „with file support” creează un meniu **File** în cadrul aplicației, ceea ce nu este cazul cu opțiunea „without file support”.

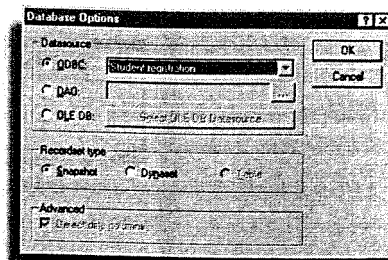
Apelați la AppWizard pentru a crea o aplicație SDI numită `UsingDB`, iar în pasul 2 selectați butonul de opțiune **Database View Without File Support**. Apoi urmați pașii de mai jos.

#### Conectarea la o sursă de date prin intermediul AppWizard

1. În caseta de dialog corespunzătoare pasului 2, selectați butonul de opțiune **Database View Without File Support** în cazul în care aplicația nu presupune existența unui meniu **File**; altfel, selectați butonul de opțiune **Database View With File Support**. Pentru exemplul nostru, selectați **Database View Without File Support**. Rețineți că această opțiune forțează crearea unei aplicații de tip SDI.
2. Efectuați un clic pe butonul **Data Source**. Este afișată caseta de dialog **Database Options**, înfățișată în figura 24.6.
3. Selectați butonul de opțiune corespunzător tipului de conectivitate dorit. Pentru exemplul nostru selectați **ODBC**.

FIGURA 24.6

Caseta de dialog Database Options.



#### Mulțimi instantaneu și dinamice de înregistrări

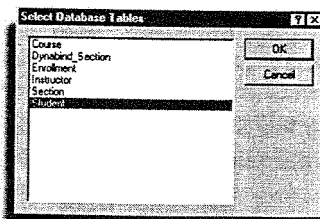
O mulțime instantaneu de înregistrări reține datele aflate la un moment dat în toate câmpurile, fără a mai ține cont de actualizări ulterioare efectuate de către alți utilizatori.

O mulțime dinamică de înregistrări reține doar câmpurile cheie. Accesarea înregistrărilor dintr-o mulțime dinamică ține cont de operațiile de actualizare și ștergere efectuate de alți utilizatori.

4. Selectați numele sursei de date din caseta combinată **Datasource**. Pentru exemplul nostru selectați **Student Registration**. Dacă această sursă de date nu apare, va trebui să o creați și să o configurați. Vedeți secțiunea de pași „Crearea și configurarea unei surse de date ODBC”, prezentată mai devreme în acest capitol.
5. Selectați tipul mulțimii de înregistrări (**Recordset Type**). Pentru exemplul nostru selectați **Snapshot**.
6. Efectuați un clic pe **OK**. Va apărea caseta de dialog **Select Database Tables**, ilustrată în figura 24.7.
7. Selectați tabela sau tabellele pentru care doriți să creați o mulțime de înregistrări. Pentru exemplul nostru selectați tabela **Student**.
8. Efectuați un clic pe **OK**. Acum ați adăugat în proiect suportul pentru baza de date; puteți continua cu pașii următori din AppWizard.

FIGURA 24.7

Caseta de dialog **Select Database Tables**.



## Conectarea la o bază de date

De conectarea la o bază de date și de accesarea informațiilor acesteia se ocupă două clase MFC care lucrează împreună: **CDatabase** și **CRecordset**. Clasa **CDatabase** răspunde de stabilirea conexiunii cu baza de date. Puteți să utilizați într-un program mai multe obiecte

**CDatabase** prin intermediul cărora să vă conectați la mai multe baze de date sau să stabiliți mai multe conexiuni cu o aceeași bază de date. Pentru că stabilirea unei conexiuni cu o bază de date este un proces relativ anevoios și care consumă multe din resursele sistemului, este de preferat să refolosiți o conexiune decât să deschideți o alta pentru fiecare mulțime de înregistrări. O soluție ar fi să încapsulați obiectul **CDatabase** în cadrul clasei document care stabilește conexiunea, creând apoi o funcție **GetDatabase()** care să întoarcă un pointer la obiectul **CDatabase**. La construcția unui obiect **CRecordset** este posibilă transmiterea ca parametru a unui pointer la un obiect **CDatabase**, astfel că mulțimea de înregistrări va folosi o conexiune existentă. Conectarea la o anumită bază de date se efectuează prin apelul funcției **CDatabase::OpenEx**. Prototipul acestei funcții este prezentat în listingul 24.1.

#### Obiecte de acces la date (DAO)

O alternativă la utilizarea claselor **CDatabase** și **CRecordset** o reprezintă **CDaoDatabase** și **CDaoRecordset**. Diferența dintre aceste clase constă în faptul că clasele DAO operează exclusiv asupra bazelor de date conduse de motorul Microsoft Jet, în timp ce clasele celelalte manipulează orice baze de date prin ODBC.

#### LISTINGUL 24.1 LST25\_1.CPP – Parametrii funcției **OpenEx**

```
1 virtual BOOL OpenEx(LPCTSTR lpstrConnectString,
2 DWORD dwOptions = 0);
```

De multe ori este precizat doar primul parametru. **lpstrString** este un pointer la un șir de caractere care conține numele sursei de date și eventuale informații adiționale necesare pentru conectarea la baza de date în cauză, cum ar fi un nume de utilizator și o parolă. Sursele de date ODBC sunt configurate prin intermediul unui program de administrare care este accesibil în cadrul panoului de control. Pentru amănunte privind configurarea unei surse de date, consultați secțiunea „Configurarea unei surse de date”, prezentată mai devreme în acest capitol. Al doilea parametru, **dwOptions**, este o mască de biți folosită pentru specificarea unei combinații între opțiunile de conectare din tabelul 24.1. Valoarea implicită 0 va deschide baza de date cu acces la scriere, biblioteca ODBC pentru cursorare nu va fi încărcată, iar dialogul pentru conectare va fi afișat doar în cazul în care șirul transmis nu conține suficiente detalii pentru stabilirea cu succes a unei conexiuni.

#### TABELUL 24.1 Valori posibile pentru parametru **dwOptions** al funcției **CDatabase::OpenEx()**

| Opțiune                | Descriere                                                                                                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>openReadOnly</b>    | Deschide sursa de date exclusiv pentru citire.                                                                                                                                                                                         |
| <b>useCursorLib</b>    | Încarcă biblioteca ODBC de tip DLL pentru cursorare. Aceasta se suprapune în parte peste funcționalitatea driverului ODBC. Spre exemplu, mulțimile dinamice nu pot fi folosite atunci când este încărcată biblioteca pentru cursorare. |
| <b>noOdbcDialog</b>    | Previne afișarea casetei de dialog ODBC pentru conectare.                                                                                                                                                                              |
| <b>forceOdbcDialog</b> | Afișează caseta de dialog ODBC pentru conectare.                                                                                                                                                                                       |

Iată un exemplu de utilizare a funcției `CDatabase::OpenEx()`:

```
CDatabase db;
db.OpenEx("ODBC;DSN=Student Registration;UID=TinyTim;PWD=
➤CASTLE", CDatabase::openReadOnly |
➤CDatabase::noOdbcDialog));
```

Mai devreme în acest capitol am menționat că este posibilă emiterea directă a unor comenzi SQL. Această facilitate este utilă în cazul în care baza de date folosită extinde standardul SQL implementat în ODBC; un exemplu în acest sens ar fi apelarea procedurilor stocate în baza de date. Emiterea directă a unei comenzi se face prin apelul funcției `ExecuteSQL` a clasei `CDatabase`. Această funcție primește un pointer la șirul de caractere care conține comanda.

## Interogarea bazei de date

Clasa `CRecordset` răspunde de accesarea datelor aflate în tabelele și coloanele unei baze de date. `CRecordset` este o clasă abstractă, ceea ce înseamnă că trebuie să derivați pe baza ei propriile clase. Interogările SQL se efectuează prin intermediul clasei `CRecordset`. Rezultatele unei interogări sunt depuse într-un buffer de înregistrări. Clasa conține un indicator de înregistrare curentă (cunoscut și sub numele de cursor) și permite parcurgerea înregistrărilor cu ajutorul funcțiilor membru `Move`, `MoveNext`, `MovePrev`, `MoveFirst` și `MoveLast`.

Clasele derivate din `CRecordset` conțin câte o variabilă membru pentru fiecare coloană (câmp) din interogare. Aceste variabile sunt actualizate cu valorile aflate în coloanele respective de fiecare dată când se efectuează un apel care schimbă înregistrarea curentă. În cadrul proiectului `UsingDB`, `AppWizard` a creat o clasă derivată din `CRecordset` care se numește `CUsingDBSet`. Definiția și implementarea acestei noi clase sunt prezentate în listingul 24.2.

### LISTINGUL 24.2 LST25\_2.CPP – Definiția clasei `CUsingDBSet`

```
1 class CUsingDBSet : public CRecordset
2 {
3 public:
4 CUsingDBSet(CDatabase* pDatabase = NULL); — ❶
5 DECLARE_DYNAMIC(CUsingDBSet)
6
7 // Field/Param Data
8 //{AFX_FIELD(CUsingDBSet, CRecordset)
9 long m_StudentID;
10 CString m_Name;
11 int m_GradYear;
12 //}AFX_FIELD — ❷
13
14 // Overrides
```

(continuare)

### LISTINGUL 24.2 Continuare

```
15 // ClassWizard generated virtual function overrides
16 //{AFX_VIRTUAL(CUsingDBSet)
17 public:
18 virtual CString GetDefaultConnect();
19 virtual CString GetDefaultSQL();
20 virtual void DoFieldExchange(CFieldExchange* pFX);
21 //}AFX_VIRTUAL
22
23 // Implementation
24 #ifdef _DEBUG
25 virtual void AssertValid() const;
26 virtual void Dump(CDumpContext& dc) const;
27 #endif
28 };
29 CUsingDBSet::CUsingDBSet(CDatabase* pdb)
30 : CRecordset(pdb)
31 {
32 //{AFX_FIELD_INIT(CUsingDBSet)
33 m_StudentID = 0;
34 m_Name = _T("");
35 m_GradYear = 0;
36 m_nFields = 3; — ❸
37 //}AFX_FIELD_INIT
38 m_nDefaultType = snapshot;
39 }
40
41 CString CUsingDBSet::GetDefaultConnect()
42 {
43 return _T("ODBC;DSN=Student registration"); — ❹
44 }
45
46 CString CUsingDBSet::GetDefaultSQL()
47 {
48 return _T("[Student]"); — ❺
49 }
```

- ❶ La construcția unui obiect `CRecordset` este posibilă transmiterea unui pointer opțional la un obiect `CDatabase` existent.
- ❷ `AppWizard` a adăugat variabile pentru fiecare câmp din mulțimea de înregistrări.
- ❸ `AppWizard` a generat inițializările pentru fiecare câmp din mulțimea de înregistrări, precum și o variabilă `m_nFields` de tip `CRecordset`.
- ❹ Este apelată de către infrastructură pentru a furniza șirul asociat conexiunii atunci când se deschide mulțimea de înregistrări.
- ❺ Este apelată de către infrastructură pentru a furniza numele tabelului atunci când se deschide mulțimea de înregistrări.



Veți vedea că în clasa document `CUsingDBDoc` există o variabilă membru `m_usingDBSet`, având tipul `CUsingDBSet`, iar în clasa reprezentare `CUsingDBView` există o variabilă membru `m_pSet` care reprezintă un pointer la obiectul mulțime de înregistrări al documentului. Acest pointer, `m_pSet`, este inițializat în funcția `OnInitUpdate`.

Definiția clasei `CUsingDBSet` a fost generată în concordanță cu opțiunile exprimate în `AppWizard`. Acolo ați selectat tabela `Student`, care conține trei coloane – `StudentID`, `Name` și `GradYear`. Puteți vedea că clasa conține trei variabile corespunzătoare (liniile 9-11). Tipurile și numele variabilelor au fost determinate pe baza informațiilor aflate chiar în baza de date. Aceste variabile sunt inițializate cu valori implicite în cadrul funcției constructor (liniile 33-35). Tot aici este inițializată și variabila `m_nFields` cu valoarea 3 (linia 36), indicând astfel numărul de coloane din mulțimea de înregistrări.

La instanțiere, constructorului `CUsingDBSet` îi poate fi transmis un pointer la un obiect `CDatabase` (linia 29); dacă acest pointer este `NULL` (așa cum este în exemplul nostru), clasa de bază `CRecordset` va crea un obiect `CDatabase` și va apela funcția `CDatabase::Open` în scopul conectării la baza de date. Înainte de apelul `CDatabase::Open` este apelată funcția virtuală `GetDefaultConnect` (linia 41), care furnizează șirul necesar pentru conectare (linia 43). Acest șir specifică obiectului bază de date sursa de date la care urmează să se conecteze.

Funcția `GetDefaultSQL` din linia 46 este apelată de către `CDatabase::Open`. Aceasta din urmă generează o instrucțiune SQL `SELECT` implicită și efectuează interogarea bazei de date. Instrucțiunea SQL generată pentru exemplul nostru va fi `"SELECT * FROM Student"`, aceasta furnizând toate liniile din tabela `Student` și stocându-le în bufferul de înregistrări. După execuția funcției `Open` puteți să vă deplasați printre înregistrările obținute și să accesați datele aflate în coloane. `CRecordset` include și o funcție `Requery` care repetă interogarea anterioară în scopul actualizării înregistrărilor din buffer fără a fi nevoie de închiderea și deschiderea din nou a bazei de date.

Există doi indicatori care arată dacă înregistrarea curentă (cursorul) se află la începutul sau la sfârșitul mulțimii de înregistrări. Acești indicatori pot fi testați prin apelul funcțiilor `IsBOF` (beginning-of-file – început de fișier) și `IsEOF` (end-of-file – sfârșit de fișier). Listingul 24.3 prezintă un exemplu de cod care parcurge fiecare înregistrare până când ajunge la sfârșitul fișierului.

#### LISTINGUL 24.3 LST25\_3.CPP – Parcurgerea unei mulțimi de înregistrări

```
1 void CMyView::OnIterateSet()
2 {
3 // ** Deschidem o multime de inregistrari
4 CMyRecordSet rs(NULL); — ❶
5 rs.Open(); — ❷
6
7 // ** Verificam daca multimea de inregistrari este vida
8 if(rs.IsBOF()) — ❸
9 return;
```

```
10
11 // ** Parcurgem fiecare inregistrare,
12 // ** apeland o functie de prelucrare
13 while(!rs.IsEOF()) — ❹
14 {
15 DoSomeFunction(&rs);
16 rs.MoveNext();
17 }
18 }
```

- ❶ Instantiază un obiect aparținând unei clase derivate din `CRecordset`.
- ❷ Deschide mulțimea de înregistrări.
- ❸ Se asigură că mulțimea de înregistrări nu este vidă (că are înregistrări).
- ❹ Bucla se repetă până când este atins sfârșitul mulțimii de înregistrări.

### Actualizarea informațiilor din baza de date

Capacitatea de interogare a unei baze de date și de extragere a informațiilor conținute este foarte utilă, dar nu vă poate duce prea departe. Trebuie, de asemenea, să puteți să adăugați, să ștergeți și să modificați înregistrări. Din fericire, `CRecordset` se ocupă de aproape tot ceea ce implică aceste operații.

#### Excepții generate de bazele de date

Este bine să știți că multe dintre funcțiile clasei `CRecordset`, printre care `Open()`, `AddNew()`, `Edit()` și `Update()`, pot să genereze o excepție de tip `CDBException` în cazul apariției unei probleme. În consecință, este să bine să încadrați apelurile acestor funcții între instrucțiuni `try/catch`. Pentru a beneficia de asistență la depanarea excepțiilor generate de bazele de date, validați opțiunea `Database tracing` a meniului `Tools, MFC Tracer`.

Adăugarea unei noi înregistrări se realizează în trei pași. Mai întâi veți apela `AddNew`, care trece mulțimea de înregistrări în modul de adăugare și stabilește valorile câmpurilor la `NULL`. Apoi veți actualiza câmpurile cu valorile pe care doriți să le stocați în noua înregistrare a bazei de date. Aceasta nu înseamnă altceva decât simpla atribuire de valori variabilelor membru ale clasei mulțime de înregistrare. În fine, veți apela `Update`, care creează noua înregistrare în cadrul bazei de date și încheie modul de adăugare. Retineți că în cazul mulțimilor de înregistrări de tip instantaneu va trebui să apelați `Requery` pentru a putea accesa noua înregistrare adăugată.

Modificarea unei înregistrări existente se efectuează în patru pași. În primul rând, veți stabili ca înregistrare curentă acea înregistrare pe care doriți să o modificați. Apoi veți apela `Edit`, care va trece mulțimea de înregistrări în modul de editare. În continuare veți actualiza valorile câmpurilor. În fine, veți apela `Update`, care înscrie valorile în baza de date, actualizează mulțimea de înregistrări și încheie modul de editare.



Ștergerea unei înregistrări se realizează într-o singură etapă. Pur și simplu, veți apela `Delete` pentru a șterge înregistrarea curentă. Aceasta va fi înlăturată din mulțimea de înregistrări și din baza de date. Este necesar să vă deplasați din dreptul înregistrării șterse deoarece încercarea de a o accesa va duce la încălcarea unei aserțiuni.

## Asocierea dintre câmpuri și coloanele tabelor din baza de date

Orice operație de actualizare sau de derulare necesită un schimb de date între bufferul de înregistrări și baza de date. Ați văzut deja că clasa mulțime de înregistrări conține câte o variabilă membru pentru fiecare coloană din interogare. Ce variabilă membru corespunde cărei coloane se specifică în cadrul funcției `DoFieldExchange`. Listingul 24.4 prezintă funcția `DoFieldExchange` a clasei `CUsingDBSet` din exemplul `UsingDB`.

Funcția `DoFieldExchange` este apelată automat atunci când este nevoie de extragerea sau de înscrierea de valori. Responsabile de schimbul datelor de diferite tipuri între coloanele dintr-o tabelă a unei baze de date și variabilele membru ale unei mulțimi de înregistrări sunt o serie de macrodefiniții oferite de MFC, prefixate cu `RFX` (`Record Field Exchange` – schimb de date între înregistrări și câmpuri). Al doilea parametru specificat într-o macrodefiniție `RFX` este numele coloanei în cauză din tabela bazei de date. Cel de-al treilea parametru este variabila membru care stochează valoarea corespunzătoare. Observați comentariile speciale (liniile 3 și 8). Nu editați și nu ștergeți aceste comentarii, deoarece sunt folosite de `ClassWizard`.

### LISTINGUL 24.4 LST25\_4.CPP – Funcția `DoFieldExchange`

```
1 void CUsingDBSet::DoFieldExchange(CFieldExchange* pFX) — ❶
2 {
3 //{AFX_FIELD_MAP(CUsingDBSet)
4 pFX->SetFieldType(CFieldExchange::outputColumn); — ❷
5 RFX_Long(pFX, _T("[StudentID]"), m_StudentID);
6 RFX_Text(pFX, _T("[Name]"), m_Name); — ❸
7 RFX_Int(pFX, _T("[GradYear]"), m_GradYear);
8 //}AFX_FIELD_MAP
9 }
```

❶ `DoFieldExchange()` este apelată de către infrastructură în scopul extragerii sau actualizării valorilor înscrise în coloanele mulțimii de înregistrări.

❷ Arată că membrii de date care urmează sunt câmpuri de ieșire, prin contrast cu parametrii de intrare.

❸ Efectuează schimbul de date între variabilele membru ale mulțimii de înregistrări și tabela bazei de date.

## Crearea și utilizarea unei reprezentări de tip înregistrare

Dacă la crearea unui proiect cu `AppWizard` ați optat pentru adăugarea suportului pentru baze de date, clasa reprezentare a aplicației va fi implicit derivată din `CRecordView`. `CRecordView` este o clasă derivată din `CFormView` și implementează o reprezentare de tip

formular. Acest formular are la bază o machetă de dialog fără chenar și fără titlu, care acoperă întreagă zonă client a ferestrei reprezentării. `CRecordView` aduce capacitatea de conectare a reprezentării de tip formular la înregistrările unei baze de date din cadrul unei clase derivate din `CRecordset`. Controalele adăugate pe macheta de dialog pot fi legate direct de variabile membru care sunt actualizate pe baza câmpurilor din baza de date. `AppWizard` adaugă, de asemenea, un meniu `Record` și butoane pe bara cu instrumente care permit utilizatorului să navigheze printre înregistrările din mulțimea de înregistrări, reprezentarea fiind actualizată automat pentru fiecare înregistrare.

## Editarea machetei reprezentării de tip înregistrare

S-ar putea ca machetele de dialog pentru reprezentările de tip înregistrare să arate diferit de alte casete de dialog, deoarece nu există margini și nici bară de titlu, dar controalele se adaugă și se editează în cadrul editorului de resurse ca în cazul oricărei alte machete de dialog. În mod normal, se adaugă câte un control pentru fiecare câmp de date care va apărea în formular. Controalele sunt atașate apoi la variabilele membru ale clasei derivate din `CRecordset`.

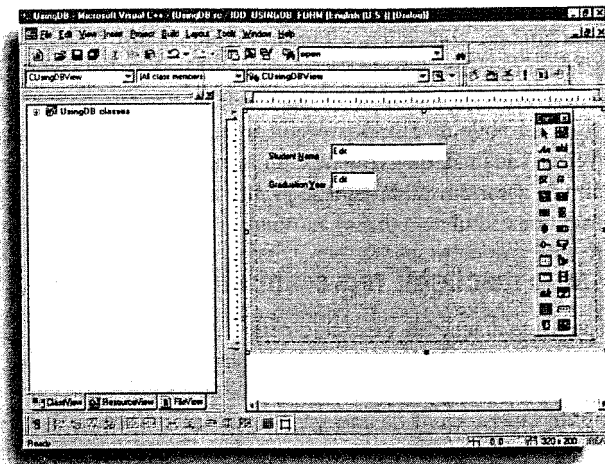
Exemplul `UsingDB` necesită adăugarea în reprezentarea de tip înregistrare a două câmpuri, unul pentru numele studentului și altul pentru anul de absolvire. Pentru proiectarea reprezentării înregistrare a aplicației `UsingDB`, urmați pașii de mai jos.

### Proiectarea machetei de dialog pentru o reprezentare de tip înregistrare

1. În cadrul paginii `ResourceView`, expandați catalogul `Dialog` și efectuați un dublu clic pe `IDD_USINGDB_FORM`. Macheta de dialog este deschisă în editorul de resurse.
2. Efectuați un clic pe eticheta **TODO: Place Form Controls Here** și apoi apăsați tasta `Delete` pentru a o înlătura.
3. Selectați controlul etichetă statică de pe bara cu instrumente `Controls`. Adăugați un astfel de control în partea stânga sus a casetei de dialog, așa cum o arată figura 24.8.
4. Specificați conținutul etichetei. Pentru exemplul nostru introduceți `Student &Name`.
5. Selectați controlul casetă de editare de pe bara cu instrumente `Controls`. Adăugați un astfel de control în partea dreaptă a etichetei `Student Name`.
6. Introduceți `IDC_NAME` în caseta combinată `ID`.
7. Selectați controlul etichetă statică de pe bara cu instrumente `Controls`. Adăugați un astfel de control sub eticheta `Student Name`.
8. Specificați conținutul etichetei. Pentru exemplul nostru introduceți `Graduation &Year`.
9. Selectați controlul casetă de editare de pe bara cu instrumente `Controls`. Adăugați un astfel de control în partea dreaptă a etichetei `Graduation Year`.
10. Introduceți `IDC_YEAR` în caseta combinată `ID`. Macheta de dialog ar trebui să arate acum ca în figura 24.8.

FIGURA 24.8

Macheta reprezentării de tip înregistrare din proiectul UsingDB.



### Atașarea controalelor de editare la câmpurile din mulțimea de înregistrări

Fiecare dintre controalele de editare aflate pe macheta de dialog a reprezentării de tip înregistrare poate fi asociat cu un câmp al bazei de date. Această asociere se realizează prin atașarea la controlul de editare a unei variabile care indică în mod direct o variabilă membru a clasei mulțime de înregistrări. Clasa reprezentare *CUUsingDBView* conține în variabila membru *m\_pSet* un pointer la obiectul *CUsingDBSet*. Pentru asocierea unui membru indicat de *m\_pSet* cu un control de editare puteți să folosiți ClassWizard.

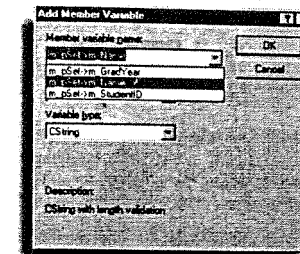
Urmați pașii de mai jos pentru a atașa casetele de editare a numelui și anului din cadrul reprezentării de tip înregistrare la câmpurile bazei de date cu studenți.

### Atașarea controalelor la câmpurile mulțimii de înregistrări cu ajutorul ClassWizard

1. În cadrul paginii ResourceView, expandați catalogul Dialog și efectuați un dublu clic pe *IDD\_USINGDB\_FORM*. Macheta de dialog este deschisă în editorul de resurse.
2. Selectați primul control casetă de editare și apoi apăsați Ctrl+W pentru a apela ClassWizard, sau selectați **ClassWizard** din meniul contextual.
3. Selectați pagina **Member Variables**.
4. Selectați identificatorul controlului de editare în cadrul casetei cu listă **Control IDs**; în cazul de față selectați *IDC\_NAME*, iar apoi efectuați un clic pe butonul **Add Variable**. Va fi afișată caseta de dialog **Add Member Variable**. Adăugarea unei variabile membru asociată unui control se poate face și prin efectuarea unui dublu clic asupra identificatorului controlului respectiv.
5. Derulați lista casetei combinate **Member Variable Name**; vor fi afișate variabilele mulțimii de înregistrări, așa cum o ilustrează figura 24.9. Selectați numele variabilei membru pe care o veți atașa la controlul de editare. Pentru exemplul nostru selectați *m\_pSet->m\_Name*.

FIGURA 24.9

Atașarea la controlul de editare a unei variabile a mulțimii de înregistrări.

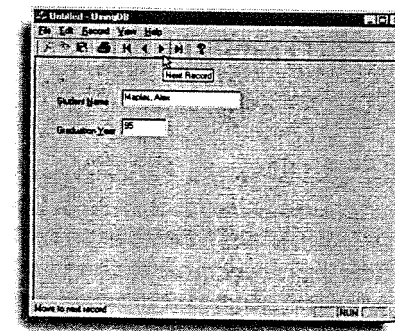


6. Selectați o valoare pentru câmpul **Category**; în cazul de față, selectați **Value**.
7. Selectați o valoare pentru câmpul **Variable Type**; în cazul de față, selectați **CString**.
8. Efectuați un clic pe **OK**.
9. Selectați *IDC\_YEAR* în caseta cu listă **Control IDs** și apoi efectuați un clic pe butonul **Add Variable**.
10. Derulați lista casetei combinate **Member Variable Name**. Selectați din această listă variabila *m\_pSet->m\_Name*.
11. Selectați o valoare pentru câmpul **Category**; în cazul de față, selectați **Value**.
12. Selectați o valoare pentru câmpul **Variable Type**; în cazul de față, selectați **int**.
13. Efectuați un clic pe **OK** pentru a închide caseta de dialog **Add Member Variable**.
14. Efectuați un clic pe **OK** pentru a închide caseta de dialog **ClassWizard**.

Acum asamblați și rulați proiectul. Aplicația ar trebui să se conecteze automat la sursa de date Student Registration, iar inițializarea reprezentării de tip înregistrare ar trebui să ducă la afișarea primei înregistrări din tabela Student. Încercați să navigați printre înregistrări. Figura 24.10 înfățișează exemplul UsingDB.

FIGURA 24.10

Exemplul UsingDB.



### VEDEȚI...

- Pentru mai multe informații privind proiectarea casetelor de dialog, vedeți pagina 54.
- Pentru amănunte suplimentare privind clasa *CFormView*, vedeți pagina 415.

# Despre programarea OLE și COM

---

Despre principiile modelului componentelor obiectuale

---

Crearea unui server propriu de automatizare OLE

---

Despre servere și containere OLE

---

## Programarea bazată pe componente

Complexitatea crescândă a dezvoltării aplicațiilor în cadrul unor medii complexe, așa cum este Windows, cu multe sale interfețe de programare a aplicațiilor (API), precum și nevoia de standardizare, de control al versiunilor, de accelerare a dezvoltării și de calcul distribuit au dus la apariția unei tehnologii numite *programare bazată pe componente*.

### COM+, viitorul COM

Următorul pas în evoluția COM va fi COM+. Microsoft promite că noul COM+ va simplifica în bună măsură scrierea codului necesar pentru COM, astfel că obiectele COM vor avea un comportament mai similar cu obiectele C++ și vor putea fi create prin intermediul cuvintelor cheie `new` și `delete` din C++. La momentul apariției acestei cărți, Visual C++ 6.0 nu include suport pentru COM+, dar este posibilă apariția ulterioară a unui kit de dezvoltare sau a unui pachet de completare pentru COM+.

Componentele sunt entități software de dimensiuni mici (asemeni instanțelor unei clase) care efectuează operații specifice prin intermediul unor interfețe bine definite. Spre deosebire de instanțele claselor, componentele nu sunt legate definitoriu de o instanță a unui program sau de un sistem gazdă. Pot fi scrise într-o multitudine de limbaje și, cu toate acestea, comunică fără probleme, prin intermediul interfețelor, cu programe și componente implementate în alte limbaje.

Microsoft a dezvoltat această tehnologie până la forma sa actuală, fiind denumită în prezent Component Object Model (COM) – modelul componentelor obiectuale. S-ar putea să întâlniți și alți termeni înrudiți, precum legarea și încapsularea obiectelor (Object Linking and Embedding – OLE) sau controale ActiveX. Trebuie să rețineți că acestea reprezintă implementări particulare ale programării COM. Propriu-zis, COM este un standard independent de limbaj și de platformă care definește modul în care obiectele pot comunica între ele prin intermediul unui protocol acceptat în comun.

Cel mai important element privind obiectele COM îl reprezintă interfețele acestora; obiectele propriu-zise nu sunt decât cutii negre care implementează o funcționalitate particulară. Dar pentru a putea să utilizați această funcționalitate, programele trebuie să respecte un contract bine definit privind transmiterea parametrilor și obținerea rezultatelor. Acest contract dintre programul client și obiect se numește *interfață*.

Întregi biblioteci API pot fi definite și proiectate în termeni de interfață. Dacă dezvoltați o aplicație client, puteți să scrieți programul astfel încât să comunice cu un server prin intermediul acestor interfețe acceptate, fără a mai conta ce aplicație server va fi utilizată. Reciproc, ați putea decide să dezvoltați componente de tip server care respectă definițiile interfețelor, comercializându-le ca alternativă viabilă la componentele altor producători.

Interfața de programare a aplicațiilor pentru mesagerie (Messaging API – MAPI) reprezintă un set de interfețe standard pentru obiectele COM. Oricine este liber să scrie obiecte COM care efectuează operații precum stocarea mesajelor, transportul mesajelor și oferirea către destinatarii mesajelor a unei agende de adrese. Microsoft Exchange este o implementare particulară a acestor componente server, dar există multe alte implementări. Codul efectiv

s-ar putea să difere enorm între diferite implementări, dar toate obiectele COM respectă aceleași specificații de interfață. Aceasta înseamnă că clienții acestor servicii, așa cum este Microsoft Outlook, pot folosi componentele compatibile MAPI ale oricărui producător în scopul trimiterii, primirii și stocării mesajelor. În mod similar, orice producător poate să ofere propriile programe client (și mulți o fac) care folosesc Exchange sau orice alte componente server MAPI, chiar fără a ști ale cui sunt componentele utilizate. Singura cerință în ceea ce privește componentele client și server este ca acestea să se apeleze reciproc prin intermediul acestor interfețe definite de comun acord, reunite sub numele de MAPI.

#### VEDEȚI...

- În legătură cu utilizarea controalelor ActiveX în propriile aplicații, vedeți pagina 186.
- În legătură cu crearea controalelor ActiveX, vedeți pagina 605.
- Pentru mai multe informații despre MAPI, vedeți pagina 675.

## Interfețe COM

O interfață este o definiție a unei mulțimi de funcții și a parametrilor acestora. Orice obiect COM dispune de cel puțin o interfață, iar în mod frecvent sunt oferite mai multe interfețe, fiecare conținând propriul său set unic de funcții.

#### Convenții pentru denumirea interfețelor

Asemeni multor lucruri, interfețele COM sunt denumite în concordanță cu funcționalitatea asociată. Printr-o convenție, însă, sunt distinse prin litera **I** care le prefixează numele. Exemple de interfețe COM care respectă această regulă sunt IUnknown, IDispatch, IMoniker sau IMessageFilter.

Un obiect COM poate fi scris în orice limbaj care are capacitatea de a suporta aceste interfețe. Unele limbaje sunt mai potrivite în acest sens decât altele. Spre exemplu, Java este foarte potrivit din cauză că fiecare obiect Java poate avea mai multe interfețe, deci se mapează natural peste un obiect COM. Obiectele COM conțin codul efectiv aflat în spatele acestor interfețe, astfel că un program care apelează o funcție declarată într-o interfață va găsi o implementare ce efectuează operația definită de acea funcție în cadrul interfeței respective.

Printr-un prea puțin ciudat joc al sorții, structura interfeței COM standard este exact aceeași cu structura tabelii de funcții virtuale din Visual C++. Aceasta înseamnă că este posibil să folosiți mecanismul tabelilor de funcții virtuale pentru a defini și implementa interfețe COM.

În mod normal, funcțiile virtuale sunt folosite ca o metodă de supradefinire în cadrul unei clasei derivate a unei funcții din clasa de bază (specificând prefixul `virtual` în declarația funcției din clasa de bază). În acest scop este declarată o tabelă de funcții virtuale (`vtable`) asociată clasei respective.

#### Tabele de funcții virtuale

Orice instanță de memorie a unui obiect C++ are atașată o tabelă de funcții virtuale (chiar dacă se poate ca unele tabele să nu conțină nici o intrare și, deci, să fie vide). Fiecare intrare din tabelă conține un pointer către codul care implementează o funcție virtuală. De fiecare dată când se apelează una dintre funcțiile virtuale ale obiectului, tabela atașată oferă adresa corectă, corespunzătoare fie funcției din clasa de bază, fie funcției din clasa derivată.

Puteți să declarați o clasă C++ care conține exclusiv funcții virtuale pure. Aceasta se numește o clasă de bază abstractă. Nu este posibilă instanțierea unei clase abstracte, dar aceste clase se pot utiliza pentru crearea unei definiții de interfață COM compatibilă cu C++, un exemplu fiind ilustrat aici:

```
class IUnknown
{
public:
 virtual HRESULT QueryInterface(REFIID riid, LPVOID FAR*
 ppvObj)=0;
 virtual ULONG AddRef()=0;
 virtual ULONG Release()=0;
}
```

Microsoft oferă astfel de definiții în C++ pentru toate interfețele obiectelor COM pe care le furnizează, deși este posibil să întâlniți funcțiile împachetate într-o macrodefiniție care va fi expandată la codul echivalent:

```
STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID FAR*
 ppvObj) PURE;
STDMETHOD_(ULONG, AddRef)(THIS) PURE;
STDMETHOD_(ULONG, Release)(THIS) PURE;
```

Toate obiectele COM trebuie să implementeze interfața IUnknown care conține cele trei metode (sau funcții) de mai sus și orice interfață trebuie să implementeze aceste trei funcții înainte de propriile funcții. Rolul acestor funcții este următorul:

- **AddRef()** – În momentul în care un client obține un pointer la o interfață, este incrementat un contor de referințe intern, contor care reprezintă numărul curent al referințelor active din cadrul clienților.
- **Release()** – În momentul în care un client eliberează un pointer la o interfață, referința internă este decrementată, obiectul fiind distrus (de regulă, de către sine) când această referință ajunge la zero.
- **QueryInterface()** – Un client care conține un pointer la o interfață poate să solicite un pointer la o altă interfață a obiectului COM, transmitând identificatorul interfeței dorite (`riid` din exemplul precedent). Dacă interfața solicitată există, `AddRef()` este apelat asupra unui pointer la această interfață, întors în cadrul pointerului `ppvObj`. Despre identificatorii de interfață (`riid`) vom discuta amănunțit în secțiunea următoare.

Pentru că toate interfețele implementează în mod garantat aceste funcții standard, puteți să scrieți un program client care solicită o instanță a unui obiect COM (care va apela automat `AddRef()`). Pentru a accesa diferitele interfețe ale obiectului veți apela la `QueryInterface()`. Odată ce ați terminat, veți apela `Release()` asupra fiecărui pointer la interfață pe care l-ați obținut pentru a permite obiectului să se distrugă pe sine (firește, mai puțin în cazul în care este folosit și de către altcineva). Aceleași funcții operează la fel pe toate obiectele COM.





(DirectDraw este un exemplu arbitrar de obiect COM și nu este important aici. Dacă doriți să vedeți cum poate fi utilizat, urmăriți secțiunea „Utilizarea interfeței DirectDraw” din capitolul 28, „Utilizarea pachetelor API și SDK”).

Pentru clasele COM se folosește un prefix similar, CLSID. Obiectul DirectDraw care implementează interfața IID\_IDirectDraw2 este cunoscut și ca CLSID\_DirectDraw.

Cu toate acestea, dedesubt, identificatorii de interfață și de clasă rămân identificatori globali unici, iar numerele efective pot fi găsite în fișierul de antet DDRAW.h, declarate astfel:

```
DEFINE_GUID(CLSID_DirectDraw,
 0xD7B70EE0, 0x4340, 0x11CF, 0xB0, 0x63, 0x00,
 0x20, 0xAF, 0xC2, 0xCD, 0x35);
DEFINE_GUID(IID_IDirectDraw2,
 0xB3A6F3E0, 0x2B43, 0x11CF, 0xA2, 0xDE, 0x00,
 0xAA, 0x00, 0xB9, 0x33, 0x56);
```

#### VEDEȚI...

- Pentru un exemplu de utilizare a DirectDraw cu CoCreateInstance() sau pentru a vedea modul de folosire a DirectDraw, vedeți pagina 657.

## Crearea de instanțe ale obiectelor COM

La crearea unei instanțe a unui obiect COM, un proces server aflat în rulare se va ocupa de inițializarea și gestionarea acesteia. Procesul server poate fi propriul program client, un DLL atașat, un executabil (.exe) aflat pe propriul calculator sau un program aflat pe un sistem la distanță. Fiecare dintre aceste ipostaze diferite se numește context, fiind prezentate toate în tabelul 25.1.

**TABELUL 25.1 Diferite contexte pentru rularea codului COM**

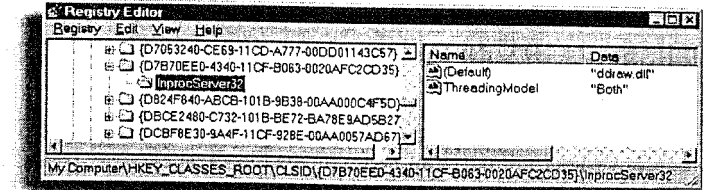
| Numele contextului  | Indicatorul asociat  | Descriere                                                                                                      |
|---------------------|----------------------|----------------------------------------------------------------------------------------------------------------|
| Server intra-proces | CLSCTX_INPROC_SERVER | Codul rulează în același proces cu programul client apelant.                                                   |
| Server local        | CLSCTX_LOCAL_SERVER  | Codul rulează într-un alt program .exe, într-un alt proces care se află pe aceeași mașină cu programul client. |
| Server la distanță  | CLSCTX_REMOTE_SERVER | Codul rulează pe o altă mașină decât programul apelant.                                                        |

Registrul din Windows este folosit pentru a asocia identificatorul global unic al unei clase (CLSID) cu un .dll sau un .exe care va crea și administra la cerere obiectul COM respectiv. Pentru toate clasele COM înregistrate în sistem există intrări corespunzătoare în HKEY\_CLASSES\_ROOT\CLSID, având ca nume identificatorii globali unici respectivi. Aici se stochează câte o cheie pentru fiecare din diferitele tipuri de servere înregistrate – InprocServer32, LocalServer32 sau RemoteServer32 (și încă altele). În cadrul fiecăreia dintre aceste chei puteți vedea numele fișierului .dll sau .exe care conține codul ce creează și administrează obiectul COM.

Spre exemplu, figura 25.2 înfățișează editorul de registru, cu ierarhia astfel expandată încât să fie afișat identificatorul de clasă al obiectului DirectDraw și administratorul său intra-proces de tip DLL, ddraw.dll.

**FIGURA 25.2**

Editorul de registru afișează identificatorul de clasă corespunzător unei intrări înregistrate.



Aceste servere implementează un obiect COM special, numit Obiect Clasă. Este un obiect asemeni oricărui alt obiect COM, dar care implementează interfața standard IClassFactory. Prin apelarea funcției CreateInstance() a acestei interfețe se poate crea o clasă de tipul dorit. De regulă, în acest scop veți folosi funcția CoCreateInstance() a bibliotecii COM pentru a crea o instanță sau CoGetObject() pentru a obține un pointer la obiectul generator de clase. Aceste generatoare de clase ar trebui, de fapt, să se numească generatoare de obiecte, deoarece construiesc obiecte, nu clase. (Programatorii sunt generatoare de clase!)

CoCreateInstance() este definită astfel:

```
STDAPI CoCreateInstance(REFCLSID rclsid,
 LPUNKNOWN pUnkOuter, DWORD dwClsContext,
 REFIID riid, LPVOID* ppv);
```

#### Crearea de obiecte pe calculatoare aflate la distanță

Pentru crearea de obiecte pe calculatoare aflate la distanță trebuie să utilizați funcția CoCreateInstanceEx(). Aceasta folosește COM distribuit (DCOM – Distributed COM), care permite comunicația între calculatoarele legate în rețea prin intermediul apelurilor de proceduri la distanță (RPC – Remote Procedure Call). Pentru înregistrarea acestor servere COM aflate la distanță, sistemele client trebuie configurate cu ajutorul utilitarului DCOMCNFG.EXE.

Primul parametru, rclsid, este o referință la un GUID reprezentând identificatorul clasei (CLSID) pe care doriți să o instanțiați. Prin cel de-al doilea parametru, pUnkOuter, se transmite de obicei valoarea NULL, acest parametru fiind utilizat doar pentru o tehnică avansată COM, numită agregare. Prin parametrul dwClsContext veți trimite în mod normal indicatorul CLSCTX\_SERVER, specificând astfel tipul de server dorit. CLSCTX\_SERVER este o combinație între indicatori din tabelul 25.1. Al patrulea parametru, riid, este o referință la identificatorul GUID al interfeței dorite pentru noul obiect COM. Ultimul parametru, ppv, este un pointer la pointerul de interfață al clientului, în care va fi plasat pointerul la interfața specificată pentru noul obiect.

Spre exemplu, pentru a crea o instanță a obiectului DirectDraw și a obține un pointer la interfața IDirectDraw2, ați putea să scrieți următoarele linii de cod:

```
IDirectDraw2* pIDirectDraw2 = NULL;
```

```
HRESULT hr = CoCreateInstance(&CLSID_DirectDraw, NULL,
 CLSCTX_ALL, &IID_IDirectDraw2, (void**) &pIDirectDraw2);
```

Dacă apelul de funcție reușește, va fi creat un nou obiect `DirectDraw`, iar `pIDirectDraw2` va indica interfața `IDirectDraw2` a noului obiect COM.

## Bibliotecile DLL de intermediere și apelurile intermediare

Atunci când propriul program client obține un pointer la o interfață a unui obiect COM, nu îl interesează unde se află obiectul efectiv. Și totuși, dacă obiectul COM rulează în afara procesului client, pointerul nu poate indica direct tabela `vtable` de funcții ale interfeței; în schimb, în spațiul propriului proces este creat un proximitar, care, din punctul de vedere al clientului, este similar cu un pointer local la interfață. În momentul în care apelezi funcții pe baza acestui pointer la interfață, propriul client și obiectul COM sunt puse în legătură cu ajutorul unui DLL terț, care se numește de intermediere, tehnica purtând numele de apel intermediat.

### Distribuirea bibliotecilor de intermediere

În cazul în care folosești apeluri intermediare între sisteme DCOM, va trebui să vă asigurați că pe calculatoarele client și gazdă există aceeași versiune a bibliotecii de intermediere.

Fiecare interfață este la rândul său înregistrată în cadrul registrului Windows, în catalogul `HKEY_CLASSES_ROOT\Interface`, sub propriul GUID. Aici sunt stocate apoi chei precum `ProxyStubClsid32`, care conțin identificatorul de clasă ce indică locația bibliotecii de intermediere care poate să efectueze apelurile intermediare ale funcțiilor interfeței. Biblioteca de intermediere are în sarcină împachetarea parametrilor într-un format independent de mașină, în scopul transmisiei. Este posibil, apoi, să folosească apelurile de procedură la distanță (RPC) pentru a apela o funcție a unui obiect COM aflat la distanță. Bibliotecile de intermediere pot fi generate automat, prin crearea definițiilor în limbajul de definire a interfețelor (IDL – Interface Definition Language) și prelucrarea acestora de către compilatorul Microsoft IDL (MIDL). Compilatorul va produce cod pentru RPC și apeluri intermediare, corespunzător unei biblioteci de intermediere.

## Versiunile interfețelor

Vechea problemă a întreținerii versiunii corecte a aplicației este rezolvată printr-o simplă regulă. Odată ce ai eliberat un obiect COM pentru a fi folosit de utilizatorii de pretutindeni, versiunile ulterioare trebuie să păstreze nealterate interfețele originale. Aceasta înseamnă că noile versiuni ale unui obiect COM trebuie să implementeze interfețe suplimentare, fără a le modifica pe cele originale. Programele client mai vechi, care nu știu să utilizeze decât interfața mai veche, vor solicita în continuare această interfață, în timp ce programele client care cunosc noile facilități pot să solicite versiunea mai nouă de interfață.

Prin convenție, numele noilor interfețe se creează prin adăugarea la numele original a numărului de versiune. Spre exemplu, interfața originală `IClassFactory` a fost dezvoltată prin adăugarea suportului pentru licențiere, așa că, pe lângă aceasta, generatoarele de clase ar

putea implementa și interfața `IClassFactory2`, cu metodele sale suplimentare. Programele client vor putea apoi să solicite `IClassFactory`, apelând la `QueryInterface()` pentru a încerca să obțină interfața `IClassFactory2` mai sofisticată. Dacă obiectul generator de clase COM nu oferă această interfață, în pointerul la interfață transmis în apelul `QueryInterface()` se înscrie valoarea `NULL`, funcția întorcând valoarea `S_FALSE`.

Pentru că tot ce contează este definiția interfeței, obiectele COM sunt libere să folosească același cod de implementare pentru funcțiile care rămân identice în versiunile vechi și nouă ale interfeței.

## Automatizarea OLE

Legarea și încapsularea obiectelor (OLE – Object Linking and Embedding) este un termen folosit inițial pentru a descrie capacitatea de a insera obiecte de diferite tipuri în documente având un tip propriu, un exemplu fiind inserarea foilor de calcul Excel în documentele Word.

Documentele care suportă operațiile de legare și încapsulare se numesc documente compuse. Pentru multe aplicații, dacă efectuați un clic pe meniul **Insert** veți vedea o opțiune **Object** corespunzătoare. Aplicațiile care oferă această opțiune lucrează cu documente compuse și vă permit să inserați obiecte pe baza unei liste de servere OLE înregistrate.

Obiectele inserate pot fi încapsulate, ceea ce înseamnă că vor fi salvate împreună cu documentul, sau legate, ceea ce înseamnă că în document va fi inserată o referință la un alt fișier (numită identificator). Acestea sunt cele două operații care au dat naștere termenului OLE original.

Dar semnificația acestui termen s-a lărgit, incluzând acum tragerea și plasarea OLE și automatizarea OLE, aceasta din urmă referindu-se la situația în care un program poate să apeleze metodele unui alt program care rulează ascuns, fără ca utilizatorul să fie conștient de această interacțiune.

### Microsoft Word: server de automatizare

Dacă aveți Microsoft Outlook și Microsoft Word instalate în sistem, ați putea observa că la compunerea unui mesaj e-mail în Outlook sunt disponibile aceleași facilități de verificare ortografică pe care le oferă Word. Acesta este un exemplu în care Word funcționează ca server de automatizare pentru Outlook. Word rulează ascuns, în fundal, fiind apelat de Outlook pentru a efectua verificarea ortografică.

## Despre interfața dispecer

Capacitatea de automatizare provine din definiția unei interfețe COM esențiale, `IDispatch`. Un obiect COM care oferă o interfață dispecer (dispatch) conține o tabelă de metode (funcții), evenimente (funcții care utilizează apeluri inverse ale clientului în scopul unei notificări a acestuia) și proprietăți (funcții care furnizează sau stabilesc valoarea unei anumite variabile a obiectului COM). Programele client pot să acceseze aceste informații dinamic (procedeul se numește legare târzie), în timpul rulării, pentru a afla detalii privind obiectul de automatizare.



**Interfețele duale**

S-ar putea să întâlniți termenul interfață duală folosit în legătură cu COM și OLE. Un obiect COM poate să implementeze o interfață duală, oferind funcțiile atât printr-o interfață directă COM, cât și printr-o interfață dispecer. Astfel, programele client pot beneficia de avantajele ambelor soluții; programele C++ rapide pot să acceseze obiectul direct prin interfața COM rapidă, iar Visual Basic și limbajele de scriptare pot să acceseze funcțiile prin interfața dispecer mai înceată.

Interfața dispecer extinde IUnknown prin următoarele patru funcții:

- **GetTypeInfoCount()** – Permite programului client să determine dacă există disponibile informații de tip.
- **TypeInfo()** – Furnizează informațiile de tip.
- **GetIDsOfNames()** – Furnizează o mulțime de indici într-un vector (numiți identificatori de dispecer) care corespund cu numele de metode, evenimente sau proprietăți solicitate.
- **Invoke()** – Apelează o funcție din vectorul dispecer, transmițându-se identificatorul de dispecer și o serie de parametri de tip VARIANT, rezultatul întors fiind tot VARIANT (structura VARIANT este prezentată în secțiunea următoare).

Această interfață dispecer este utilizată intens în OLE, fiind legată de controale ActiveX, documente OLE și scripturi Visual Basic. O astfel de metodă de căutare a funcțiilor la momentul execuției pe baza unei tabele interne este mult mai înceată decât apelarea directă a metodelor prin interfață, însă aduce o anumită flexibilitate. Se pot construi biblioteci întregi de informații de tip, sub forma unor fișiere .tlb; acestea permit Visual C++ să genereze rapid clase schelet (numite drivere de dispecer) care oferă metodele corespunzătoare, arătând exact ca o versiune locală a obiectului de automatizare. Aceste clase schelet operează cu parametri adecvați, pe care îi transformă într-un vector de elemente VARIANT, utilizând funcția **Invoke()** pentru a apela obiectul de automatizare OLE propriu-zis.

**Utilizarea varianțelor**

Din cauză că aceeași funcție **Invoke()** este apelată pentru multe funcții diferite de automatizare OLE, trebuie să existe o cale de transmitere a diferitelor tipuri de parametri către diversele funcții apelate. În acest scop se folosește o structură **DISPPARAMS**, care indică un vector de structuri VARIANT. O structură VARIANT conține o uniune C++ cuprinzând multe tipuri diferite de date care pot fi prelucrate de OLE. Există, de asemenea, un indicator de tip (**VARTYPE**), numit **vt**, care specifică tipul reprezentat. Tabelul 25.2 prezintă câteva din multe tipuri de date care pot fi stocate într-o structură VARIANT. Aceste structuri pot să rețină și pointeri la obiectele suportate, fiind necesară prefixarea cu **p** a numelui corespunzător obiectului (de pildă, **p1Val**) și specificarea indicatorului **VT\_BYREF** în cadrul indicatorului de tip (de exemplu, **VT\_I4|VT\_BYREF**).

**Transmiterea vectorilor de varianți**

În apelurile funcțiilor OLE se pot transmite vectori întregi de varianți prin intermediul clasei **COleSafeArray**. Această clasă vă permite să definiți tipul elementelor, numărul acestora și dimensiunea vectorului, astfel că puteți să transmiteți blocuri mari de date într-un singur apel. Consultați documentația standard oferită de Microsoft pentru a afla mai multe amănunte privind utilizarea clasei **COleSafeArray**.

**TABELUL 25.2 Câteva dintre tipurile de date permise în structurile VARIANT**

| Tipul de date | Nume     | Indicator de tip | Descriere                                                                                                        |
|---------------|----------|------------------|------------------------------------------------------------------------------------------------------------------|
| unsigned char | bVal     | VT_UI1           | Valoare fără semn pe un octet                                                                                    |
| short         | iVal     | VT_I2            | Valoare cu semn pe doi octeți                                                                                    |
| long          | lVal     | VT_I4            | Valoare cu semn pe patru octeți                                                                                  |
| float         | fltVal   | VT_R4            | Valoare în virgulă mobilă pe patru octeți                                                                        |
| double        | dblVal   | VT_R8            | Valoare în virgulă mobilă pe opt octeți                                                                          |
| BOOL          | boolVal  | VT_BOOL          | Valoare TRUE sau FALSE pe patru octeți                                                                           |
| SCODE         | scode    | VT_ERROR         | Cod de eroare COM                                                                                                |
| DATE          | date     | VT_DATE          | Valoare în virgulă mobilă care reține o dată într-un format compatibil cu <b>COleDateTime</b>                    |
| BSTR          | bstrVal  | VT_BSTR          | Valoare de tip șir de caractere compatibilă cu Visual Basic, care poate fi convertită la și de la <b>CString</b> |
| IUnknown      | punkVal  | VT_UNKNOWN       | Pointer la o interfață IUnknown                                                                                  |
| IDispatch     | pdispVal | VT_DISPATCH      | Pointer la o interfață IDispatch                                                                                 |

Tipurile de date care pot fi reținute într-un variant sunt independente de limbaj și pot fi recunoscute de orice limbaj compatibil cu OLE. Pentru a efectua conversia între **BSTR** (tip de șir din Visual Basic) și un obiect **CString** MFC puteți să folosiți funcțiile **AllocSysString()** și **SetSysString()** ale clasei **CString**, care alocă spațiu pentru un șir **BSTR** și, respectiv, stabilesc conținutul unui șir **BSTR** existent pe baza conținutului obiectului **CString**. Transformarea inversă poate fi efectuată prin conversiile de tip (**char\***) și (**const char\***), care furnizează un șir cu terminator nul pe baza unui șir **BSTR**.

**COleDateTime** poate să accepte și transmiterea directă a unei structuri VARIANT ca parametru pentru constructor, dar este posibilă și specificarea unei valori DATE obținute prin intermediul conversiei de tip (**DATE**).

Varianții nu sunt folosiți numai de către interfața dispecer; ei pot fi întâlniți frecvent în OLE, reprezentând o soluție utilă pentru stocarea și transferul diferitelor tipuri de date.

**Crearea unui server de automatizare**

Multe aplicații, printre care Word, Excel sau chiar Developer Studio, sunt servere de automatizare. Acestea pun la dispoziția aplicațiilor client o interfață dispecer care poate fi

utilizată pentru apelarea funcțiilor proprii, oferind servicii specializate precum verificarea ortografică.

VB Scripting (Visual Basic Scripting), instrumentul utilizat pentru crearea macrocomenzilor, folosește aceste metode în scopul creării și manipulării documentelor din cadrul serverelor de automatizare. De fiecare dată când scrieți o macrocomandă pentru unul dintre aceste programe, folosiți de fapt o versiune redusă de Visual Basic care permite apelarea funcțiilor din interfața dispecer a serverului de automatizare respectiv.

Serverele de automatizare sunt asociate de obicei cu un nume inteligibil care poate fi utilizat pentru determinarea identificatorilor de clasă corespunzători. Aceste nume sunt reținute în registrul sistem, în HKEY\_CLASSES\_ROOT, conținând sub-chei ce specifică identificatorul de clasă al serverului de automatizare respectiv. Spre exemplu, pentru Microsoft Visual Studio există intrarea de mai jos, care-i asociază identificatorul de clasă corespunzător:

```
HKEY_CLASSES_ROOT\MSDEV.APPLICATION\CLSID =
{FB7FDAE2-89B8-11CF-9BE8-00A0C90A632C}
```

Clientul poate apoi să folosească funcția `CLSIDFromString()`, transmitând o versiune inteligibilă a identificatorului de clasă (`MSDEV.APPLICATION`) și un pointer la o structură `CLSID` în care va fi înscrisă forma numerică a acestui identificator (`{FB7FDAE2-89B8-11CF-9BE8-00A0C90A632C}`).

#### Vizualizarea tabelului cu obiectele aflate în rulare

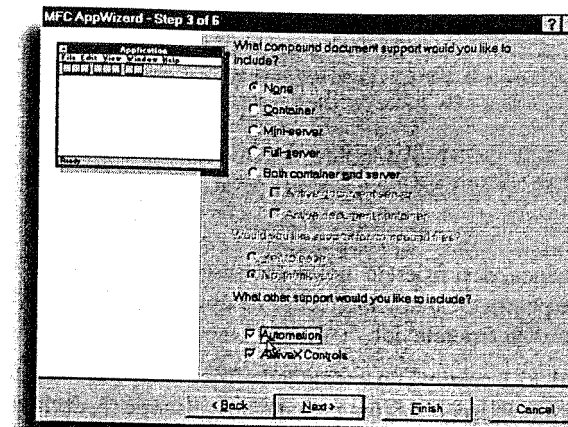
Serverele de automatizare lansate în execuție se înregistrează de multe ori într-o tabelă a obiectelor aflate în rulare, permițând astfel clienților să se conecteze la instanța care rulează. Această înregistrare se efectuează prin intermediul unui „moniker” – de fapt, un identificator pentru programe, documente compuse sau fragmente de date. Obiectele în rulare înregistrate la un moment dat pot fi vizualizate cu ajutorul utilitarului... Microsoft Visual Studio\Common\IROTview.exe. Odată lansat, IROTview.exe va afișa obiectele aflate în rulare (și care s-au înregistrat). Visual Studio își înregistrează propria clasă sub identificatorul `{FB7FDAE2-89B8-11CF-9BE8-00A0C90A632C}`. Dacă deschideți un document Microsoft Word, veți vedea că numele documentului este înregistrat în tabela cu obiectele în rulare.

În continuare, programul client poate să apeleze `CoCreateInstance()`, pentru a crea o instanță invizibilă de `Developer Studio`, care rulează în fundal, solicitând un pointer la interfața dispecer. Programele .exe ale serverelor de automatizare pot fi lansate cu parametrul de linie de comandă `/Automation`, împiedicând astfel afișarea normală în fereastră. Atunci când rulează în acest mod, interacțiunea cu serverul trebuie să aibă loc printr-o interfață dispecer. Word, Excel și toate celelalte servere de automatizare se comportă în acest fel, permițând programelor client să beneficieze de funcționalitatea lor fără ca utilizatorul să fie conștient de comunicația între programe.

Infrastructura unei aplicații server de automatizare poate fi creată prin intermediul AppWizard, validând opțiunea `Automation` prezentă în pasul 3 din MFC AppWizard (vedeți figura 25.3).

FIGURA 25.3

Adăugarea suportului pentru automatizare prin intermediul AppWizard.



Atunci când creați infrastructura unui proiect cu suport pentru automatizare, veți observa că apar o serie de fișiere adiționale. Unul dintre acestea este un fișier .reg care conține intrările de registru necesare pentru noul server de automatizare. Acest fișier este necesar doar dacă aveți în vedere instalarea serverului de automatizare pe alte calculatoare prin intermediul unui program de instalare. De asemenea, serverul de automatizare va crea automat intrările necesare în cadrul registrului atunci când este lansat în mod normal, fără parametrul `/Automation`.

AppWizard generează automat un GUID pentru clasa documentului, acest identificator putând fi găsit în noul fișier .reg. De exemplu, dacă creați infrastructura pentru un server de automatizare numit `Autoserver`, în fișierul .reg vor fi generate următoarele intrări de registru:

```
HKEY_CLASSES_ROOT\Autoserver.Document = Autose Document
HKEY_CLASSES_ROOT\Autoserver.Document\CLSID =
{D11ED783-CFD8-11D1-931D-444553540000}
HKEY_CLASSES_ROOT\CLSID\{D11ED783-CFD8-11D1-931D-4445535400
→00} =
Autose Document
HKEY_CLASSES_ROOT\CLSID\{D11ED783-CFD8-11D1-931D-4445535400
→00}\ProgId = Autoserver.Document
HKEY_CLASSES_ROOT\CLSID\{D11ED783-CFD8-11D1-931D-4445535400
→00}\LocalServer32 = AUTOSERVER.EXE
```

Celălalt fișier adițional este un fișier .odl. Aici se află descrierea în limbaj de definiție a obiectului, care va fi folosită la generarea unei biblioteci de tipuri pentru noul server de automatizare. Atunci când adăugați noi metode sau proprietăți în cadrul serverului de automatizare, în acest fișier vor fi înscrise intrări care descriu parametrii noilor funcții. Nu trebuie să vă preocupați de întreținerea sau compilarea fișierului, deoarece `Class Wizard` asigură în întregime întreținerea lui, iar compilarea se efectuează automat la asamblarea proiectului.

**Bibliotecile de tip, MktypLib și MIDL**

Înainte, bibliotecile de tipuri erau compilate pe baza fișierelor scrise în limbajul de definiție a obiectelor (.odl), prin intermediul utilitarului MktypLib. Fișierele similare scrise în limbajul de definiție a interfețelor (.idl) erau utilizate pentru definirea specificațiilor interfețelor, dar nu puteau să creeze biblioteci de tipuri. Într-un timp, Microsoft a reglementat situația, dezvoltând fișierele .idl astfel încât să poată specifica și informațiile pentru biblioteca de tipuri. Noul compilator Microsoft pentru IDL (MIDL) poate să compileze atât fișierele .idl, cât și fișierele .odl (MktypLib devenind inutil).

De asemenea, deși fișierele .odl se folosesc în continuare, acestea nu sunt neapărat necesare pentru crearea bibliotecilor de tipuri, deoarece specificațiile corespunzătoare pot fi declarate în fișierele .idl.

Listingul 25.1 prezintă un exemplu de fișier .odl pentru serverul de automatizare Autoserver. Există o singură metodă, `SquareRoot()`, declarată în linia 26 în cadrul interfeței dispecer `IAutoserver`, descrisă între liniile 13 și 28.

Exemplul Autoserver prezentat aici implementează metoda `SquareRoot()`, care întoarce o valoare `double` reprezentând rădăcina pătrată a unei valori `double` de intrare.

**LISTINGUL 25.1 LST26\_1.ODL – Descrierea în limbaj de definiție a obiectului corespunzător unui server de automatizare numit Autoserver și conținând o singură metodă**

```
1 //autoserver.odl:type library source for autoserver.exe
2
3 // This file will be processed by the MIDL compiler to
4 // produce the type library (autoserver.tlb)
5
6 [uuid(D11ED784-CFD8-11D1-931D-444553540000),
 version(1.0)]
7 library Autoserver
8 {
9 importlib("stdole32.tlb");
10
11 // Primary dispatch interface for CAutoserverDoc
12
13 [uuid(D11ED784-CFD8-11D1-931D-444553540000)]
14 dispinterface IAutoserver
15 {
16 properties:
17 // NOTE - ClassWizard will maintain property information
18 // Use extreme caution when editing this section.
19 //{{{AFX_ODL_PROP(CAutoserverDoc) — ❶
20 //}}}AFX_ODL_proprietaea
21
22 methods:
23 // NOTE - ClassWizard will maintain method information
```

```
24 // Use extreme caution when editing this section.
25 //{{{AFX_ODL_METHOD(CAutoserverDoc) — ❷
26 [id(1)] double SquareRoot(double dInputVal);
27 //}}}AFX_ODL_METHOD
28 };
29
30 // Class information for CAutoserverDoc
31
32 [uuid(D11ED784-CFD8-11D1-931D-444553540000)]
33 coclass Document
34 {
35 [default] dispinterface IAutoserver;
36 };
37 //{{{AFX_APPEND_ODL}}
38 //}}}AFX_APPEND_ODL}}
39 };
```

❶ ClassWizard adaugă aici proprietățile OLE, sub formă de metode get/set.

❷ ClassWizard adaugă aici metodele OLE.

❸ Această secțiune declară documentul ca o clasă cu o interfață dispecer.

De asemenea, AppWizard adaugă cod sursă în cadrul infrastructurii standard, asigurând suportul pentru automatizarea OLE. Funcția `InitInstance()` din clasa aplicației (`CMyServerApp`) trebuie să inițializeze bibliotecile OLE și COM, apelând în acest sens funcția `AfxOleInit()`.

Veți vedea, totodată, o linie suplimentară care conectează macheta documentului aplicației la o nouă variabilă membru de tip `COleTemplateServer`, numită `m_server`:

```
m_server.ConnectTemplate(clsid, pDocTemplate, TRUE);
```

Acest server de machete este derivat din `COleObjectFactory`, implementarea specifică OLE a generatorului de clase. Aplicațiile client vor folosi acest generator de clase pentru a crea o nouă instanță a obiectului document al serverului de automatizare, fie cu ajutorul `CoCreateInstance()`, fie prin `CoGetClassObject()` și `CreateInstance()`.

Dacă aplicația este apelată ca server de automatizare de către o aplicație client, va înregistra obiectele de automatizare OLE ca fiind pregătite pentru utilizare, prin următoarele linii adiționale din `InitInstance()`:

```
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
 COleTemplateServer::RegisterAll();
 return TRUE;
}
```

**Parametrii din linia de comandă pentru încapsulare și automatizare**

Atunci când o aplicație client apelează funcțiile unei aplicații server de automatizare .Exe, serverul (cum este Word) trebuie să ruleze nevăzut, în fundal. OLE realizează acest lucru lansând programul server cu parametrul /Automation specificat în linia de comandă. Funcția ParseCommandLine(), apelată din funcția InitInstance() a serverului, activează indicatorul m\_bRunAutomated al obiectului CCommandLineInfo. Prin urmare, aplicația poate să testeze acest indicator, verificând dacă programul este apelat ca server de automatizare dintr-o aplicație client sau dacă este rulat ca aplicație independentă, de sine stătătoare. Dacă programul este lansat cu parametrul /Embedded specificat în linia de comandă, aplicația știe că este folosită pentru gestionarea unui obiect încapsulat într-un *document compus*.

Altfel, serverul de automatizare va fi rulat ca o aplicație obișnuită, iar liniile de mai jos vor înscrise în registrul Windows intrările necesare:

```
m_server.UpdateRegistry(OAT_DISPATCH_OBJECT);
CObjectFactory::UpdateRegistryAll();
```

Dacă privești clasa document generată de AppWizard, vei vedea că și aici a fost adăugat cod prin intermediul căruia documentul devine un obiect de automatizare.

Identificatorul pentru interfața dispecer a documentului este declarată ca valoare constantă GUID statică:

```
static const IID IID_IAutoserver =
{ 0xd11ed785, 0xcfd8, 0x11d1, { 0x93, 0x1d, 0x44, 0x45,
 0x53, 0x54, 0x0, 0x0 } };
```

Interfața dispecer propriu-zisă este implementată cu ajutorul unor macrodefiniții generate de AppWizard:

```
BEGIN_INTERFACE_MAP(CAutoserverDoc, CDocument)
INTERFACE_PART(CAutoserverDoc, IID_IAutoserver, Dispatch)
END_INTERFACE_MAP()
```

Metodele din harta de dispecer sunt adăugate automat prin intermediul unei alte serii de macrodefiniții, create tocmai pentru descrierea acestei hărți. De exemplu, o nouă metodă SquareRoot() adăugată în documentul Autoserver ar fi descrisă astfel:

```
BEGIN_DISPATCH_MAP(CAutoserverDoc, CDocument)
//{{AFX_DISPATCH_MAP(CAutoserverDoc)
DISP_FUNCTION(CAutoserverDoc, "SquareRoot", SquareRoot,
 VT_R8, VTS_R8)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

Șirul "SquareRoot" furnizează interfeței dispecer nume metodei, iar indicatorii VT\_R8 și VTS\_R8 specifică tipul parametrului și al rezultatului întors (adică, valori în virgulă mobilă de tip double).

**Parametrii macrodefiniției DISP\_FUNCTION**

Primul parametru transmis macrodefiniției DISP\_FUNCTION reprezintă numele clasei care implementează funcția. Al doilea parametru precizează numele extern al funcției – acesta este numele pe care-l vor găsi clienții de automatizare atunci când inspectează interfața dispecer a serverului. Cel de-al treilea parametru furnizează numele funcției de implementare. Al patrulea parametru specifică tipul valorii VARIANT pe care o întoarce funcția. Ultimul parametru constă dintr-o listă a tipurilor de valori VARIANT care vor fi transmise funcției, separate prin spații. Acești parametri sunt definiți (în AFXDISP.H) ca un șir de valori caracter binare care sunt convertite ulterior în indici ce specifică tipul valorilor VARIANT.

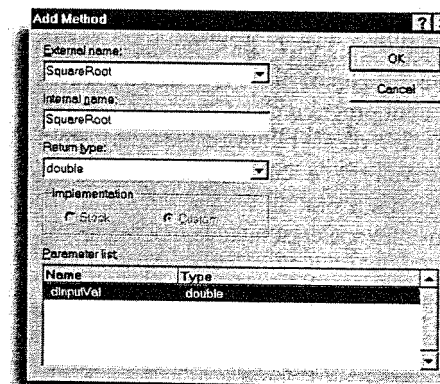
Nu interveniți asupra acestor macrodefiniții; ClassWizard se ocupă de gestionarea lor atunci când adăugați noi metode de automatizare.

**Adăugarea cu ClassWizard a unei metode pentru serverul de automatizare**

1. Apăsați Ctrl+W sau efectuați un clic pe meniul **View** și selectați opțiunea **ClassWizard** pentru a apela ClassWizard.
2. Selectați pagina **Automation** și asigurați-vă că în caseta combinată **Class Name** este selectată clasa documentului.
3. Efectuați un clic pe butonul **Add Method** pentru a adăuga noua metodă.
4. Introduceți numele extern al metodei, cel care va fi prezentat aplicațiilor client, așa cum este SquareRoot pentru serverul CAutoserverDoc, ca în figura 25.4.
5. Dedesubt veți vedea numele funcției interne care va fi apelată. Acest nume poate fi modificat, dacă este nevoie, însă de regulă se acceptă numele implicit.
6. Selectați din caseta combinată derulantă **Return Type** tipul valorii întoarse de noua metodă. (Pentru exemplul Autoserver, acest tip ar trebui să fie double.)

FIGURA 25.4

Adăugarea unei noi metode în cadrul unui obiect server de automatizare prin intermediul ClassWizard.



7. Introduceți în caseta **Parameter List** numele primului parametru care va fi transmis aplicației la producerea evenimentului (dInputVal, pentru cazul de față).

8. Selectați tipul noului parametru în cadrul casetei combinate alături de acestuia, în coloana **Type**. (Pentru exemplul Autoserver, acest tip ar trebui să fie `double`.)
9. Repetați pașii 6 și 7 pentru fiecare parametru transmis de la aplicația client la noua funcție a serverului de automatizare.
10. Efectuați un clic pe **OK** pentru a adăuga noua metodă, care ar trebui să apară în lista numelor externe afișată în pagina **Automation** din **ClassWizard**.
11. Efectuați un clic pe butonul **Edit Code** pentru a începe să editați codul noii metode de automatizare.

Odată adăugată noua metodă, puteți să adăugați codul de implementare care va efectua operația dorită pentru serverul de automatizare.

Metoda `SquareRoot()` din exemplul Autoserver va întoarce pur și simplu rădăcina pătrată a valorii transmise, apelând funcția matematică `sqrt()`:

```
#include "math.h"
double CAutoserverDoc::SquareRoot(double dInputVal)
{
 return sqrt(dInputVal);
}
```

Pașii urmați în **ClassWizard** și propriul cod de implementare reprezintă tot ce trebuie pentru adăugarea unei noi metode de automatizare. **ClassWizard** va adăuga automat intrările necesare în harta de dispecer și în fișierul `.odl`, generând informațiile de tip necesare pentru serverul de automatizare OLE care corespund noii metode.

Dacă asamblați noul server de automatizare, veți vedea că fișierul `.odl` este compilat împreună cu codul sursă de implementare. În directorul destinație **Debug** (sau **Release**) va apărea un fișier adițional `.tlb` care conține definițiile de bibliotecă pentru informațiile de tip corespunzătoare noilor metode dispecer.

Această bibliotecă de tipuri poate fi utilizată pentru a genera o clasă driver de dispecer OLE în cadrul unui program client (așa cum veți vedea în secțiunea care urmează). Ați putea să distribuiți fișierele bibliotecă de tipuri altor programatori care doresc să apeleze serverul dumneavoastră de automatizare. De exemplu, puteți să descărcați bibliotecile de tipuri pentru Word din situl de Web al Microsoft ([www.microsoft.com](http://www.microsoft.com)) și să dezvoltați programe client care apelează metode din Word.

#### Bibliotecile de tipuri

Rețineți că, în timp ce bibliotecile statice (`.lib`) și bibliotecile cu legare dinamică (`.dll`) conțin cod executabil, bibliotecile de tipuri conțin doar definițiile pentru tipurile parametrilor și ale valorilor întoarse corespunzătoare metodelor unui server de automatizare OLE. Codul executabil se află chiar în serverul de automatizare. Spre exemplu, dacă descărcați biblioteca de tipuri pentru Microsoft Word, veți avea doar informațiile necesare pentru a transmite parametri funcțiilor oferite de Word și pentru a ști valorile întoarse. Apelarea efectivă a acestor funcții necesită existența unei versiuni de Word instalate.

## Crearea unui client de automatizare

Puteți să creați instanțe ale serverelor de automatizare și să apeleți metodele acestora din orice tip de aplicație client. Primul lucru necesar este să inițializați bibliotecile OLE. Aceasta se face prin simpla adăugare a unui apel `AfxOleInit()` în funcția

`InitInstance()` a aplicației:

```
BOOL CAutoclientApp::InitInstance()
{
 AfxOleInit();
```

În continuare, puteți să apeleți la **ClassWizard** pentru a crea o clasă driver de dispecer OLE pe baza unei biblioteci de tipuri (fișier `.tlb`), înlesnind astfel apelul metodelor oferite de serverul de automatizare.

### Adăugarea cu **ClassWizard** a unei clase driver de dispecer OLE pe baza unei biblioteci de tipuri

1. Apăsăți **Ctrl+W** sau efectuați un clic pe meniul **View** și selectați opțiunea **ClassWizard** pentru a apela **ClassWizard**.
2. Selectați pagina **Automation**.
3. Efectuați un clic pe butonul **Add Class** și selectați din meniul derulant opțiunea **From a Type Library**.
4. Localizați biblioteca de tipuri dorită, care se poate afla într-un fișier `.tlb`, `.olb` sau `.dll`.
5. Odată găsit fișierul dorit, efectuați un dublu clic pe acesta sau selectați **Open** pentru a afișa caseta de dialog **Confirm Classes**.
6. Puteți să modificați numele implicite ale claselor și ale fișierelor de implementare care vor fi generate pentru fiecare clasă, selectând clasa în cauză și modificând conținutul casetelor de editare **Class Name**, **Header File** sau **Implementation File**.
7. Efectuați un clic pe **OK** pentru a efectua importul claselor selectate; acestea vor fi adăugate în cadrul proiectului.

Ați putea să utilizați biblioteca de tipuri creată în cadrul exemplului de server de mai devreme (`autoserver.tlb`) pentru a genera fișierele `autoserver.cpp` și `autoserver.h` conținând clasa driver de dispecer `IAutoserver`. Această clasă conține pseudo-implementări de funcții care apelează funcția `Invoke()` (printr-o funcție ajutoare) în scopul apelării pe baza identificatorului din tabelă a funcțiilor corespunzătoare din interfața dispecer a serverului de automatizare.

#### Numele claselor driver de dispecer

Este neplăcut faptul că clasa driver de dispecer OLE generată de **ClassWizard** are numele implicit prefixat cu litera **I**. Aceasta face ca numele clasei să poată fi confundat cu definiția unei interfețe COM (și nu este cazul). Clasa driver de dispecer este doar o încapsulare comodă pentru apelul metodei `Invoke()` a interfeței `IDispatch` a unui obiect server de automatizare. Nu trebuie să confundați numele claselor driver de dispecer cu numele interfețelor COM.

De exemplu, funcția `SquareRoot()` din `Autoserver` este implementată în clasa driver de dispecer ca aici:

```
double IAutoserver::SquareRoot(double dInputVal)
{
 double result;
 static BYTE parms[] = VTS_R8;
 InvokeHelper(0x1, DISPATCH_METHOD, VT_R8,
 (void*)&result;
 return result;
}
```

Primul parametru (0x1) din apelul de mai sus al funcției `InvokeHelper()` reprezintă identificatorul funcției `SquareRoot()` în cadrul tabelii de funcții a interfeței dispecer a serverului de automatizare. Cel de-al doilea parametru este un indicator care arată că este vorba despre un apel de metodă, nu de eveniment sau proprietate. Al treilea parametru specifică tipul valorii întoarse, aceasta urmând a fi înscrisă în `result`. Al cincilea parametru, `parms`, precizează tipurile valorilor transmise ca parametri. Parametrii care urmează reprezintă valorile de transmis, acestea fiind împachetate de `InvokeHelper()` într-un vector de parametri `DISPPARAMS` destinat funcției `Invoke()`.

Înainte de a putea apela aceste funcții ale serverului de automatizare, va trebui să obținem un pointer la interfața dispecer a obiectului. Dacă aruncați o privire asupra fișierului de antet care conține definiția clasei driver de dispecer, veți vedea că aceasta este derivată din clasa `COleDispatchDriver`. Această clasă conține funcții membru care ajută la crearea pointerului la interfața dispecer.

Astfel, puteți să apelați metoda `CreateDispatch()` a clasei driver de dispecer, transmițând șirul inteligibil asociat cu clasa dorită (de exemplu, "Autoserver.document"). Funcția `CreateDispatch()` va determina identificatorul de clasă corespunzător din registrul sistem și va crea o instanță a clasei serverului de automatizare solicitat. În caz de succes, valoarea întoarsă este `TRUE` și devine posibilă apelarea metodelor din clasa driver.

#### Intrările din registru corespunzătoare obiectului de automatizare

Obiectele de automatizare se află în cadrul cheii `HKEY_CLASSES_ROOT` a registrului sistem. Pentru fiecare obiect există o cheie `CLSID` care conține un șir de caractere ce reprezintă valoarea GUID pe 128 de biți care identifică fiecare clasă de automatizare.

Listingul 25.2 înfățișează o secvență de cod care creează o instanță a noii clase driver de dispecer și apoi creează o interfață dispecer, apelând una dintre metodele serverului de automatizare.

Puteți să creați o aplicație de tip dialog, numită `Autoclient`, pentru a testa implementarea serverului de automatizare. Adăugați secvența de cod din listingul 25.2 la finalul funcției `OnInitDialog()` a aplicației dialog (în fișierul de implementare `autoclientDlg.cpp`). Directiva `#include` din linia a doua trebuie adăugată la începutul fișierului `autoclientDlg.cpp`.

Dacă asamblați și rulați apoi aplicația, veți vedea caseta de mesaj ilustrată în figura 25.2; programul server de automatizare este încărcat automat în fundal și pune la dispoziție funcția de extragere a rădăcinii pătrate.

#### LISTINGUL 25.2 LST26\_2.CPP – Inițializarea și apelarea unei funcții a serverului de automatizare prin intermediul unei clase driver de dispecer

```
1 // Includem fișierul de antet pentru clasa driver de dispecer
2 #include "autoserver.h"
3
4 ///.. Declarația funcției
5
6 // ** Inițializăm clasa driver de dispecer
7 IAutoserver IAutoserver;
8
9 // ** Cream o instanță a serverului și accesăm
10 // ** interfața dispecer a acestuia
11 if (IAutoserver.CreateDispatch("Autoserver.document")) — ❶
12 {
13 // ** Apelăm metoda SquareRoot a programului server
14 double dNumber = 36.0;
15 double dRoot = IAutoserver.SquareRoot(dNumber);
16
17 // ** Afisăm rezultatul
18 CString strMsg;
19 strMsg.Format("Root of %f = %f\n", dNumber, dRoot);
20 AfxMessageBox(strMsg);
21 }
```

❶ `CreateDispatch()` determină valoarea `CLSID` pentru numele specificat și apelează `CoCreateInstance()`, solicitând interfața dispecer.

Linia 2 a listingului 25.2 are ca efect includerea fișierului de antet care conține definiția clasei driver de dispecer (creată pe baza bibliotecii de tipuri). Liniile de la 6 la 21 ilustrează o secvență de cod care apelează serverul de automatizare. În linia 7 este declarată o instanță a clasei driver de dispecer. În linia 11 este apelat membrul `CreateDispatch()`, ceea ce duce la crearea unei instanțe a obiectului document al serverului de automatizare și obținerea (în membrul `m_lDispatch`) a unui pointer la interfața dispecer. Dacă apelul reușește, în linia 15 este apelată metoda `SquareRoot()`, valoarea întoarsă de aceasta, `dRoot`, fiind afișată în linia 20 printr-o casetă de mesaj, așa cum se vede în figura 25.5.

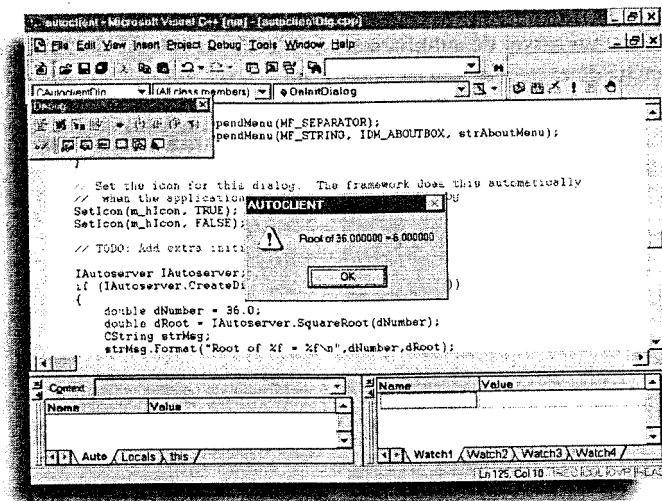
În momentul în care domeniul de existență al obiectului `IAutoserver` se încheie, interfața dispecer va fi eliberată automat de către funcția destructor a clasei.

Alternativ, eliberarea interfeței dispecer se poate face prin apelul funcției membru `ReleaseDispatch()` a driverului de dispecer. Sunt disponibile, de asemenea, metodele `AttachDispatch()` și `DetachDispatch()`, care realizează atașarea și detașarea dinamică a unui pointer la o interfață dispecer care este deja conectat la un obiect server de automatizare.



FIGURA 25.5

Funcționarea exemplului de automatizare OLE.



## Containere, servere și mini-servere OLE

Un server OLE este o aplicație care poate fi lansată în cadrul unei ferestre a unei aplicații container. De exemplu, foile de calcul din Excel pot fi inserate și editate în Word. În acest caz, Word reprezintă aplicația container OLE, iar Excel este aplicația server OLE.

### Documente active

S-ar putea să întâlniți termenul de „document activ” folosit în legătură cu serverele și containerele OLE. Este vorba despre o tehnologie relativ nouă, legată de ActiveX, care dezvoltă arhitectura container/server OLE pentru a permite obiectelor încapsulate să câștige controlul asupra întregii zone client a containerului (nu doar asupra unui mic cadru) și să manipuleze direct fereastra, meniurile și barele cu instrumente ale acestuia. Informații suplimentare despre documentele active pot fi găsite în documentația standard oferită de Microsoft, în legătură cu IOleDocument și interfețele sale înrudite.

Aplicațiile pot fi concomitent containere OLE și servere OLE, caz în care pot juca ambele roluri. Word și Excel sunt exemple potrivite în acest sens, pentru că puteți să rulați Excel și să inserați documente Word și viceversa.

Un server complet poate să ruleze ca aplicație de sine stătătoare sau ca obiect inserat, în timp ce mini-serverele pot fi lansate doar din cadrul unui program container.

Containerele OLE includ opțiunea de meniu **Insert/Object**. Aplicația WordPad din Windows este un client OLE. Puteți să folosiți această aplicație pentru a testa diferite servere OLE, efectuând un clic pe meniul **Insert** și selectând opțiunea **Object**. Veți vedea o listă cu serverele OLE înregistrate care pot fi inserate în documentul WordPad.

La inserarea unui obiect server OLE, acesta poate fi editat în interiorul unei suprafețe rectangulare care este mărginită de un chenar gros, hașurat, de culoare neagră, numit cadru

local. Operația se numește activare locală și este inițiată prin transmiterea unor indicatori, numiți verbe, de la container către server. Atunci când nu este activ, cadrul dispare și imaginea afișată este ultima imagine generată atunci când acesta era activ (stocată și reprodusă pe baza unui context dispozitiv de tip meta-fișier). Atunci când este activ, cadrul este afișat, iar la meniul propriu al containerului sunt adăugate elemente de meniu ale obiectului server.

Datele documentului încapsulat pot fi serializate de către documentul container prin intermediul funcțiilor obișnuite de serializare. Documentele server primesc de la documentul container un obiect CArchive care le permite serializarea fiecărui obiect de date specific.

### Documentele compuse: încapsularea și legarea

Dacă un document încapsulat este stocat integral în cadrul documentului compus al aplicației client, obiectele legate sunt stocate sub forma unor simpli identificatori. Un astfel de identificator (moniker) este un mic obiect care identifică în mod unic locația datelor propriu-zise și modul de afișare a acestora în cadrul aplicației client.

Infrastructura standard pentru servere OLE și containere OLE poate fi creată cu ajutorul AppWizard; cu toate acestea, între container și server există o cooperare complexă care trebuie implementată pentru a asigura funcționarea corespunzătoare a ambelor părți.

Infrastructura standard pentru serverele OLE conține două clase adiționale care asigură suportul pentru editarea locală. Una dintre aceste clase este o clasă server derivată din `COleServerItem`. Aceasta este folosită pentru reprezentarea obiectului încapsulat în cadrul containerului și include o funcție `OnDraw()` în scopul generării reprezentării obiectului la cererea containerului. Există, de asemenea, o funcție `OnDoVerb()` care gestionează cererile emise de aplicația container pentru afișarea, deschiderea sau ascunderea obiectului încapsulat. În plus, containerul poate să comunice cu serverul printr-o interfață `IOleObject` oferită de către server.

Cealaltă clasă este o fereastră cadru locală, fiind derivată din `COleIPFrameWnd`. Această clasă este folosită pentru implementarea barelor cu instrumente și a opțiunilor de meniu ale serverului în locul celor ale containerului atunci când serverul este activat.

Infrastructura generată pentru un container conține o singură clasă suplimentară, o clasă derivată din `COleClientItem`. Această nouă clasă conține o serie de funcții care ajută la crearea și întreținerea obiectelor legate sau încapsulate în cadrul unui document compus. Sunt implementate, de asemenea, legătura pe partea de container cu obiectul server, comunicarea prin intermediul verbelor cu ajutorul unei funcții `DoVerb()`, precum și mai multe funcții care au rolul de a gestiona poziționarea corectă a obiectului server și starea sa de activare.

Prin personalizarea celor două clase adăugate pe partea de server și a clasei suplimentare a containerului puteți să implementați suport pentru operații destul de complexe de editare locală la rularea într-un cadru în fereastra unei aplicații container. Nici serverul și nici containerul nu au nevoie să cunoască tipul obiectului care încapsulează sau al celui conținut. Atât timp cât obiectele server și client respectă același standard, containerul și serverul pot fi integrate perfect pentru a da utilizatorului impresia unei aplicații unitare.



## Crearea controalelor ActiveX

Crearea propriilor controale ActiveX

Implementarea propriilor pagini de proprietăți ale controalelor

Crearea fișierelor pentru distribuția și înregistrarea controalelor

Testarea controlului cu ajutorului containerului de test

## Crearea unei infrastructuri ActiveX prin intermediul ActiveX Control Wizard

Există un asistent AppWizard special, destinat anume generării de infrastructuri pentru controalele ActiveX. Acest asistent vă ajută prin generarea de cod, licențierea și adăugarea suportului pentru asistență pentru noul control. Pentru apelarea sa, efectuați un clic pe meniul **File** și selectați **New** pentru a deschide caseta de dialog **New**. Accesați pagina **Projects** și veți vedea o opțiune numită **MFC ActiveX ControlWizard**. Selectați această opțiune pentru a crea un nou control numit **Rotary** (în câmpul **Project Name**).

## Controalele ActiveX și Active Template Library

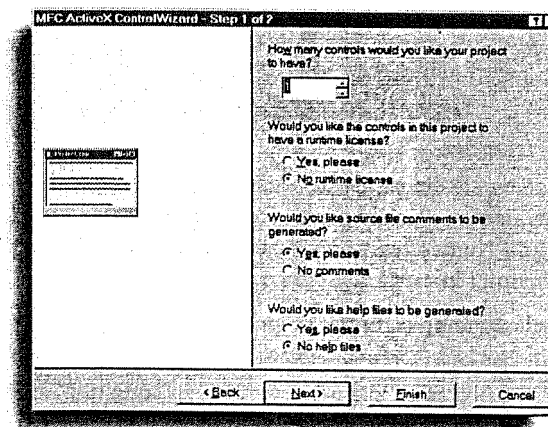
Capitolul de față se oprește asupra creării controalelor ActiveX cu ajutorul claselor de bază Microsoft (MFC), dar aceste controale pot fi dezvoltate și pe baza bibliotecii de machete active (ATL – Active Template Library). Controalele ActiveX create cu ATL sunt în general mai rapide și consumă mai puțină memorie decât controalele ActiveX bazate pe MFC. Oricum, folosirea împreună a claselor MFC și ATL nu este deloc la îndemână. ATL reprezintă un subiect vast și complex, depășind scopul urmărit de această carte. Dacă aveți nevoie de controale ActiveX foarte rapide, sofisticate și cu consum scăzut de memorie, ar trebui să luați în considerație varianta ATL și opțiunea ATL COM AppWizard, în locul soluției MFC.

## Specificarea numărului de controale, a licențierii și a suportului pentru asistență

Primul pas din MFC ActiveX ControlWizard (înfățișat în figura 26.1) vă permite să adăugați diferite tipuri de suport în cadrul noului proiect de tip control ActiveX.

FIGURA 26.1

MFC ActiveX ControlWizard  
– Step 1 of 2.



- Prima întrebare vă cere să specificați numărul de controale din cadrul proiectului. Dacă alegeți un număr mai mare decât valoarea 1 implicită, vor fi generate clase control și clase de pagini de proprietăți pentru fiecare control.

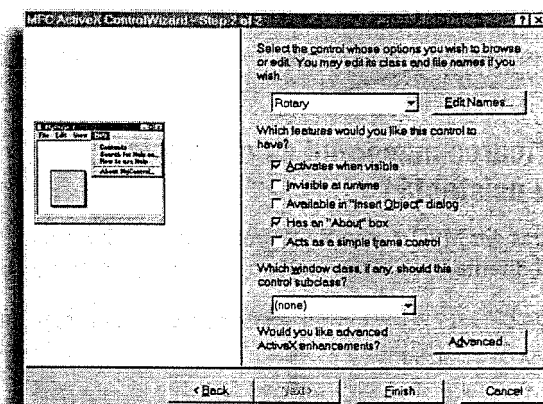
- Următoarea întrebare este dacă doriți licențierea controalelor din proiect. Dacă răspundeți pozitiv la această întrebare, clasele vor fi dotate cu funcții de verificare a licenței și va fi generat un fișier de licență .lic implicit. În consecință, controalele vor putea fi distribuite numai împreună cu un fișier de licență adecvat.
- Următoarea întrebare, privind generarea de comentarii în cadrul codului sursă, permite adăugarea în cod a unor comentarii similare celor generate de AppWizard în aplicațiile SDI și MDI.
- Ultima întrebare din acest pas privește generarea fișierelor de asistență. Dacă răspundeți pozitiv, în cadrul proiectului vor fi generate fișiere de asistență contextuală.

## Specificarea numelor de clase și a opțiunilor de utilizare

În pasul 2 aveți posibilitatea de a specifica o serie întreagă de opțiuni care privesc modul în care controlul apare și este utilizat (vedeți figura 26.2).

FIGURA 26.2

MFC ActiveX ControlWizard  
– Step 2 of 2.



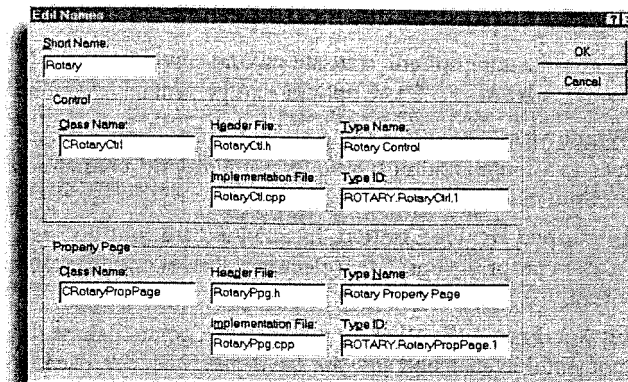
Caseta combinată din partea superioară și butonul **Edit Names** vă permit să modificați numele de fișier și de clasă pentru controlul selectat în casetă. La efectuarea unui clic pe butonul **Edit Names** veți vedea lista cu numele fișierelor și claselor care vor fi generate pentru controlul în cauză, așa cum o arată figura 26.3.

Există o clasă pentru controlul propriu-zis (CRotaryCtrl) și o clasă pentru pagina de proprietăți asociată (CRotaryPropPage). Deoarece controalele ActiveX sunt obiecte COM/OLE, în partea dreaptă a casetei de dialog Edit Names puteți vedea numele (**Type Name**) și identificatorul (**Type ID**) de tip asociate. Acestea sunt numele care vor fi utilizate în Visual Basic (Scripting) sau în alte limbaje în care doriți să folosiți controlul ActiveX. Acum este momentul să schimbați, eventual, nume de clase, de fișiere și de tip.

Dacă închideți caseta de dialog și reveniți în fereastra MFC ActiveX ControlWizard – Step 2 of 2 (figura 26.2), puteți vedea lista opțiunilor disponibile pentru control.

FIGURA 26.3

Caseta de dialog Edit Names vă permite să modificați numele implicite ale claselor și fișierelor generate.



- **Activates when Visible.** Această opțiune este în mod normal selectată și determină containerul controlului să activeze automat controlul în momentul în care devine vizibil.
- **Invisible at Runtime.** Este posibil să scrieți controale ActiveX care nu afișează nimic, dar care pot fi inserate într-un container OLE cu scopul de a adăuga o anumită funcționalitate.
- **Available in "Insert Object" Dialog.** Dacă doriți ca utilizatorii să poată insera controlul într-un fișier Word, Excel sau HTML, validați această casetă.
- **Has an "About" Box.** Adaugă în proiect o casetă de dialog About.
- **Acts as a Simple Frame Control.** Configurează informațiile de stare care permit acestui control să conțină alte controale din Windows.

## Subclasarea controalelor existente în scopul moștenirii funcționalității

Caseta combinată din partea inferioară a ferestrei poate fi utilizată pentru alegerea unui control pe baza căruia să subclasați (derivați) controlul ActiveX. Aceasta vă permite să dezvoltați funcționalitatea unui control obișnuit din Windows. Spre exemplu, dacă doriți să îmbunătățiți bara de derulare standard, puteți să subclasați controlul pe baza acesteia pentru a-i moșteni funcționalitatea pe care o oferă.

### Tratarea mesajelor reflectate ale controlului

Una dintre sarcinile principale pe care o presupune subclasarea unui control existent din Windows este tratarea mesajelor pe care în mod normal controlul le trimite înspre fereastra sa părinte. Pentru a împiedica trimiterea acestor mesaje direct către fereastra container, clasa controlului derivat (COleControl) creează o fereastră al cărei unic scop este să joace rolul de fereastră părinte a controlului, primind astfel mesajele trimise de acesta. Va trebui apoi să interceptați mesajele relevante generate de control, provocând acolo unde este necesar un eveniment ActiveX din partea propriului control.

## Utilizarea opțiunilor avansate pentru controlul ActiveX

Pentru afișarea unor opțiuni avansate privind controlul ActiveX puteți să efectuați un clic pe butonul **Advanced**. Aceste opțiuni sunt prezentate în continuare:

- **Windowless Activation.** Selectați această opțiune astfel încât controlul să folosească fereastra containerului, și nu o fereastră proprie, ceea ce aduce un spor de viteză, dar necesită mai mult efort de implementare.
- **Unclipped Device Context.** În cazul în care controlul este civilizat și nu desenează în afara propriei suprafețe, selectarea acestei opțiuni va aduce și ea un mic spor de viteză.
- **Flicker-Free Activation.** În mod normal, controalele se desenează pe ele însele în momentul în care sunt activate sau dezactivate, dar, dacă între cele două stări nu există diferențe de aspect, puteți selecta această opțiune pentru a preveni cererile de redesenare.
- **Mouse Pointer Notifications When Inactive.** Selectarea acestei opțiuni permite prelucrarea mesajelor generate de deplasarea mouse-ului chiar și atunci când controlul nu este activ.
- **Optimized Drawing Code.** Unele containere permit un mod de operare care conservă indicatorii către obiectele GDI folosite în controale. Puteți să testați această facilitare și, în cazul în care există, puteți scrie cod de desenare care să refolească acești indicatori.
- **Loads Properties Asynchronously.** În cazul în care controlul este încapsulat într-o pagină de Web, puteți implementa o descărcare asincronă care să permită rularea interfeței cu utilizatorul cât mai repede posibil. Puteți să ignorați imaginile și celelalte proprietăți care se descarcă mai greu și prezintă mai puțină importanță până când sunt încărcate proprietățile esențiale.

### VEDEȚI...

➤ Pentru mai multe informații privind utilizarea controalelor ActiveX, vedeți pagina 186.

## Implementarea controlului

Sarcina unui control ActiveX este să permită interacțiunea cu utilizatorul. În acest scop va trebui, de regulă, să prezinte o interfață grafică cu utilizatorul, să trateze evenimentele provocate de tastatură, de mouse sau de alte dispozitive de intrare, și să aibă capacitatea de a transforma aceste evenimente într-o reprezentare pe care utilizatorul o va înțelege și pe care aplicația care folosește controlul va putea să o acceseze și să o utilizeze. De obicei există proprietăți pe care aplicația le poate stabili pentru a modifica aspectul sau comportamentul implicit al controlului. Prin urmare, controlul este o mini-aplicație de sine stătătoare și trebuie privit din această perspectivă. Dacă ați scris aplicații care tratează mesajele din Windows, veți recunoaște arhitectura MFC standard care se folosește la implementarea controalelor ActiveX.

### Clasele dintr-un proiect de control ActiveX

Atunci când creați cu AppWizard un control ActiveX bazat pe MFC, veți vedea că pentru implementarea infrastructurii controlului ActiveX sunt create mai multe clase. Există o clasă aplicație care implementează `InitInstance()` și `ExitInstance()` și este derivată din `COleControlModule`. O altă clasă răspunde de paginile de proprietăți ale controlului, fiind derivată din `COlePropertyPages`. În fine, clasa principală a controlului este derivată din `COleControl`, aici fiind reunite diferitele aspecte care privesc controlul ActiveX. De asemenea, remarcăți că `COleControl` este derivată din `CWnd`, oferind funcționalitatea obișnuită a unei ferestre. Aceasta ajută la crearea mediului familiar asociat cu funcțiile membru și de tratare ale clasei `CWnd`, pe care le-ați întâlnit deja în casete de dialog și reprezentări.

Cu rol de exemplu didactic, ați putea să creați un control potențiomtru circular care să se comporte asemeni unui buton de reglare a volumului sau a luminozității. La crearea acestui proiect, numit Rotary, acceptați opțiunile implicite propuse de ControlWizard.

## Desenarea controlului

Codul care se ocupă de desenare vă va părea destul de familiar în cazul în care ați implementat operații de desenare prin intermediul claselor MFC, în aplicații SDI/MDI și de tip dialog. Funcția `OnDraw()` a clasei principale a controlului, `CRotaryCtrl`, este apelată pentru desenarea controlului. Totuși, parametrii primiți diferă puțin de cei ai funcției `OnDraw()` cunoscute. Primul parametru este pointerul obișnuit la contextul dispozitiv. Următorul parametru este o referință la un obiect dreptunghi care reprezintă cadrul controlului, iar ultimul este o referință la un obiect dreptunghi care reprezintă zona invalidată ce trebuie regenerată.

Pentru a reprezenta acest potențiomtru circular, veți converti poziția în care se efectuează clic sau în care a fost tras mouse-ul într-un unghi care să reprezinte înclinarea unui vector ce pornește din centru și are vârful în poziția mouse-ului. Un cerc mic va fi desenat la 90% din lățime și înălțime față de centrul controlului, iar un cerc mai mare va reprezenta corpul potențiometrului. O linie trasată între centru și cercul mic va indica poziția curentă a potențiometrului, asemănător unei roțițe de control al volumului sau luminozității.

### LISTINGUL 26.1 LST27\_1.CPP – Implementarea funcției `OnDraw()` a controlului ActiveX

```
1 void CRotaryCtrl::OnDraw(CDC* pDC,
2 const CRect& rcBounds, const CRect& rcInvalid)
3 {
4 // ** Cream pensule pentru fundal si pentru desenare
5 CBrush brForeGnd(RGB(0,255,0));
6 CBrush brBackGnd(TranslateColor(AmbientBackColor())); — 1
7
8 // ** Desenam fundalul controlului
9 pDC->FillRect(rcBounds, &brBackGnd);
10
```

(continuare)

## LISTINGUL 26.1 Continuare

```

11 CBrush* pOldBrush = pdc->SelectObject(&brForeGnd);
12
13 // ** Calculam pozitia relativa si centrul controlului
14 CPoint ptRelative = m_ptClicked - rcBounds.TopLeft();
15 CPoint ptMid(rcBounds.Width()/2,rcBounds.Height()/2);
16
17 // ** Determinam distantele fata de centru
18 double dRelX = ptRelative.x - ptMid.x;
19 double dRelY = ptRelative.y - ptMid.y;
20
21 // ** Apelam la trigonometrie pentru a calcula unghiul
22 double dAngle = atan2(dRelY,dRelX); — ②
23 double dRadX = (double)ptMid.x * 0.9;
24 double dRadY = (double)ptMid.y * 0.9;
25
26 // ** Calculam un punct de pe raza directoare
27 int nXPos = ptMid.x + (int)(cos(dAngle) * dRadX);
28 int nYPos = ptMid.y + (int)(sin(dAngle) * dRadY);
29
30 // ** Calculam pozitia indicatorului de pozitie
31 CPoint ptKnob=CPoint(nXPos,nYPos)+rcBounds.TopLeft();
32
33 // ** Cream un dreptunghi si desenam cercul indicator
34 CRect rcPoint(ptKnob-CSize(4,4),CSize(8,8));
35 pdc->Ellipse(rcPoint); — ③
36
37 // ** Desenam corpul circular al potentiometrului
38 pdc->Ellipse(ptMid.x-(int)dRadX,ptMid.y-(int)dRadY,
39 ptMid.x+(int)dRadX,ptMid.y+(int)dRadY);
40
41 // ** Trasam o linie intre centru si indicator
42 pdc->MoveTo(ptMid);
43 pdc->LineTo(ptKnob);
44
45 pdc->SelectObject(pOldBrush);
46 }

```

① Este folosită culoarea de fundal ambiantă a containerului, astfel încât culoarea de fundal a controlului se confundă cu fereastra container.

② Pentru determinarea unghiului față de poziția mouse-ului se folosește o relație trigonometrică.

③ O elipsă reprezintă corpul circular al controlului, iar o elipsă mai mică reprezintă indicatorul de poziție.

Implementarea `OnDraw()` din listingul 26.1 are ca efect desenarea controlului. Liniile 5 și 6 creează pensulele pentru afișare și pentru fundal, fiind alese culorile RGB(0,255,0)

(verde) și, respectiv, `AmbientBackColor()`. Există și o funcție pereche, `AmbientForeColor()`. Aceste funcții furnizează două proprietăți ambientale comune tuturor controalelor. La folosirea lor în cadrul unui container, aceste proprietăți moștenesc valorile containerului, dar ele pot fi stabilite prin cod de către aplicația container sau pot fi puse la dispoziție în pagina de proprietăți asociată. Am ales explicit culoarea verde pentru afișare pentru că funcția `AmbientForeColor()` nu este întotdeauna disponibilă. Culoarea de fundal este folosită în linia 9 pentru generarea fundalului; într-o situație concretă, probabil că veți optimiza această operație astfel încât corpul potențiometrului să nu mai fie acoperit la desenarea fundalului, reducând efectul de pălpăire.

## Stabilirea proprietăților ambientale

Există o serie de proprietăți ambientale comune tuturor controalelor ActiveX. Aceste proprietăți permit controlului să se integreze în aplicația sa părinte. Cum este imposibil să știți în ce tip de aplicație va fi utilizat controlul dumneavoastră, este bine să implementați cât mai multe astfel de proprietăți ambientale.

Pentru că dreptunghiurile transmise conțin coordonate ale controlului exprimate în raport cu containerul, coordonatele poziției `m_ptClicked` a mouse-ului trebuie calculate relativ la colțul stânga sus al controlului, ca în linia 14. Linia 15 calculează coordonatele relative ale centrului controlului.

Liniile de la 17 la 28 apelează la relații trigonometrice pentru a calcula unghiul de înclinare al drepte care unește centrul controlului și poziția mouse-ului, transformând acest unghi înapoi în coordonatele unui punct aflat la 90% de la centrul controlului, aici fiind afișat indicatorul de poziție (linia 35). Pentru utilizarea funcțiilor trigonometrice trebuie să includeți un fișier de antet care conține declarații de funcții matematice, adăugând linia următoare la începutul fișierului `RotaryCtrl.cpp`:

```
#include "math.h"
```

În linia 38 este desenat cercul mai mare care reprezintă corpul potențiometrului, fiind suprapus peste cercul indicator de poziție astfel încât acesta rămâne o neregularitate a circumferinței.

Utilizarea fișierului de antet `math.h`

Deși toate funcțiile matematice sunt legate la codul dumneavoastră în cadrul bibliotecii de execuție C++ standard, va trebui să includeți fișierul de antet `math.h` pentru a furniza compilatorului prototipurile de funcții. Aveți la dispoziție un număr mare de funcții matematice uzuale, pentru trigonometrie, calculul logaritmic, operații de rotunjire și o serie de conversii pentru tipurile de date în virgulă mobilă.

Liniile 42 și 43 răspund de trasarea unei linii din centru până la această neregularitate, reprezentând astfel poziția unghiulară a controlului potențiometru.

Trebuie, de asemenea, să adăugați poziția `m_ptClicked` ca variabilă membru `CPoint` a clasei `CRotaryCtrl`. În acest scop puteți apela caseta de dialog `Add Member Variable`, accesibilă din meniul contextual al clasei `CRotaryCtrl` din cadrul paginii `ClassView`. În definiția clasei `CRotaryCtrl` va apărea, astfel, variabila următoare:

```
CPoint m_ptClicked;
```

## VEDEȚI...

- Pentru amănunte legate de trasarea liniilor și a cercurilor, vedeți pagina 349.
- Dacă aveți nevoie de asistență în ceea ce privește adăugarea variabilei membru `m_ptClicked` de tip `CPoint`, vedeți pagina 175.

## Tratarea acțiunilor utilizatorului și a evenimentelor generate

Poziția mouse-ului, `m_ptClicked`, este folosită în listingul 26.1 pentru a calcula un unghi pe baza acțiunii utilizatorului. Prin urmare, trebuie să scrieți codul care actualizează această variabilă membru atunci când utilizatorul efectuează un clic sau deplasează mouse-ul ținând apăsat butonul stâng. Pentru aceasta puteți adăuga o funcție de tratare a mesajului `WM_LBUTTONDOWN`, care va reține poziția clicului și va captura mouse-ul, așa cum o arată listingul 26.2.

### LISTINGUL 26.2 LST27\_2.CPP – Implementarea funcției de tratare `OnLButtonDown()`, care reține poziția mouse-ului și inițiază capturarea acestuia

```
1 void CRotaryCtrl::OnLButtonDown(UINT nFlags, CPoint point)
2 {
3 // ** Apelam implementarea din clasa de baza
4 CControl::OnLButtonDown(nFlags, point);
5
6 // ** Capturam mouse-ul
7 SetCapture();
8
9 // ** Stocam poziția clicului
10 m_ptClicked = point; ❶
11
12 // ** Redesenăm controlul
13 InvalidateControl();
14 }
```

- ❶ Poziția clicului este stocată, fiind folosită ulterior, în `OnDraw()`, la desenarea indicatorului de poziție al controlului.

În listingul 26.2, mouse-ul este capturat prin apelul funcției `SetCapture()` din linia 7. Noua variabilă membru `m_ptClicked` a clasei este folosită în linia 10 pentru a reține poziția în care s-a efectuat clic. În fine, apelul `InvalidateControl()` este similar cu `Invalidate()` în cadrul unei ferestre obișnuite, determinând redesenarea întregului control. Este posibil, de asemenea, să transmiteți în apelul `InvalidateControl()` un dreptunghi care să indice zona de control care trebuie redesenată.

În momentul în care utilizatorul apasă butonul mouse-ului, poziția controlului trebuie rotită astfel încât să corespundă cu poziția cursorului pe toată durata deplasării acestuia. În acest scop va trebui să adăugați o funcție de tratare a mesajului `WM_MOUSEMOVE`. Apelați încă o dată la ClassWizard pentru a crea o funcție de tratare `OnMouseMove()`, implementarea necesară fiind prezentată în listingul 26.3.

### LISTINGUL 26.3 LST27\_3.CPP – Implementarea funcției de tratare `OnMouseMove()`, care stochează și actualizează poziția controlului

```
1 void CRotaryCtrl::OnMouseMove(UINT nFlags, CPoint point)
2 {
3 // ** Verificam dacă butonul stâng al mouse-ului este apăsat
4 if (nFlags & MK_LBUTTON)
5 {
6 // ** Retinem poziția mouse-ului
7 m_ptClicked = point;
8
9 // ** Redesenăm controlul
10 InvalidateControl(); ❶
11 }
12
13 // ** Apelăm implementarea din clasa de baza
14 CControl::OnMouseMove(nFlags, point);
15 }
```

- ❶ Dacă butonul stâng al mouse-ului este apăsat, poziția controlului este actualizată și controlul se redesenează odată cu deplasarea mouse-ului.

Linia 4 din listingul 26.3 verifică dacă butonul stâng al mouse-ului este apăsat, testând prezența indicatorului `MK_LBUTTON` în cadrul parametrului `nFlags`. În caz că da, poziția cursorului este actualizată și controlul este redesenat pentru a reflecta deplasarea.

În fine, va trebui să tratați mesajul `WM_LBUTTONUP` în cadrul unei funcții `OnLButtonUp()` generate cu ClassWizard, pentru a încheia capturarea mouse-ului. Odată creată această funcție, nu trebuie decât să adăugați apelul `ReleaseCapture()` înainte de apelul implementării din clasa de bază, ca aici:

```
ReleaseCapture();
```

## VEDEȚI...

- Pentru o explicație amănunțită privind evenimentele generate de mouse și capturarea mouse-ului, vedeți pagina 169.

## Un rapid test parțial al controlului

În acest moment există suficient cod pentru a fi posibilă asamblarea controlului. Dacă asamblați acum controlul veți observa că se efectuează o operație adițională:

```
Registering ActiveX Control...
```

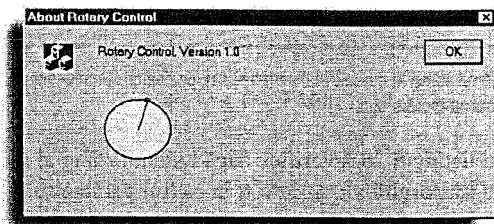
Este vorba despre înregistrarea controlului prin intermediul programului `regsvr32.exe`, care apelează de fapt o funcție globală a controlului, `DllRegisterServer(void)`, funcție care efectuează operația propriu-zisă de înregistrare. Această funcție și perechea sa pentru înlăturarea înregistrării, `DllUnregisterServer(void)`, pot fi găsite în fișierul de implementare `Rotary.cpp`.

Controlul nu poate fi rulat direct, pentru că este destinat utilizării într-o aplicație container. Dar dacă vă amintiți capitolul 9, „Utilizarea controalelor ActiveX”, controalele ActiveX pot fi adăugate în casetele de dialog. Cum noul control este înregistrat, puteți folosi editorul de resurse pentru a plasa acest control în propria casetă de dialog About (IDD\_ABOUTBOX\_ROTARY). În acest scop, deschideți mai întâi macheta de dialog IDD\_ABOUTBOX\_ROTARY în cadrul editorului de resurse (efectuând un dublu clic pe IDD\_ABOUTBOX\_ROTARY din lista resurselor de tip dialog din secțiunea spațiului de lucru al proiectului). Apoi puteți să adăugați controlul în cadrul casetei About în maniera cunoscută, efectuând un clic cu butonul drept pe suprafața machetei și selectând opțiunea **Insert ActiveX Control**.

În continuare, puteți să apăsați Ctrl+T sau să efectuați un clic pe meniul **Layout** și să selectați **Test** pentru a testa noul control (vedeți figura 26.4). În acest test din cadrul editorului de resurse veți putea să efectuați clic pe noul control potențiometru circular și să deplasați mouse-ul pentru a modifica în mod corespunzător poziția acestuia.

FIGURA 26.4

Un test rapid și mai puțin ortodox, cu ajutorul propriei casete de dialog About a proiectului.



Odată ce ați încheiat testul, nu uitați să ștergeți controlul din caseta de dialog About, deoarece nu are ce căuta acolo și poate cauza tot felul de probleme neplăcute.

#### Salvarea fișierelor proiectului înainte de testarea controalelor ActiveX

Nu este nici o problemă dacă testați un control ActiveX în cadrul editorului de resurse de tip casetă de dialog, dar nu uitați să salvați în prealabil toate fișierele proiectului, deoarece erori în implementarea controlului pot cauza blocări serioase sau probleme cu resursele. Din fericire, Visual C++ salvează fișierele înainte de compilare, ceea ce înseamnă că sunteți asigurat imediat după compilare.

#### VEDEȚI...

➤ În legătură cu adăugarea unui control ActiveX într-o casetă de dialog, vedeți pagina 190.

### Implementarea generării de evenimente

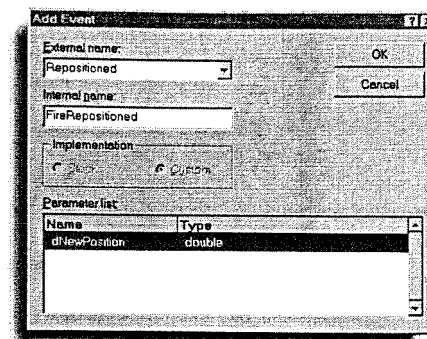
Un control trebuie să informeze aplicația posesoare atunci când utilizatorul îi modifică starea într-un fel sau altul. Controalele ActiveX nu fac excepție, iar în cadrul controlului Rotary ar trebui să adăugați o funcție de tratare care să informeze aplicația posesoare în legătură cu modificarea poziției. Această tehnică se numește *generare de evenimente*, iar pentru adăugarea unui eveniment ActiveX puteți să utilizați ClassWizard, urmând pașii de pe pagina următoare.

### Adăugarea unui eveniment al unui control ActiveX cu ajutorul ClassWizard

1. Apăsați Ctrl+W sau efectuați clic pe meniul **View** și selectați opțiunea **ClassWizard** pentru a apela ClassWizard.
2. Selectați pagina **ActiveX Events**.
3. Efectuați un clic pe butonul **Add Event** pentru a adăuga noul eveniment.
4. În caseta combinată **External Name**, specificați numele extern al evenimentului, cel care va fi vizibil pentru aplicația posesoare, ca de pildă **Repositioned** în cazul controlului potențiometru (vedeți figura 26.5).

FIGURA 26.5

Adăugarea unui eveniment ActiveX cu ajutorul casetei de dialog Add Event.



5. În caseta **Internal Name** va apărea numele intern, al funcției apelate (cea care generează evenimentul); acest nume poate fi modificat la nevoie, dar de regulă se acceptă numele implicit.
6. Introduceți în caseta **Parameter List** numele primului parametru (**dNewPosition**) care va fi transmis aplicației posesoare de către evenimentul generat.
7. Selectați tipul acestui parametru în cadrul casetei combinate alăturată noului parametru, în coloana **Type**.
8. Repetați pașii 6 și 7 pentru fiecare parametru care trebuie transmis aplicației posesoare în legătură cu producerea noului eveniment.
9. Efectuați un clic pe **OK** pentru a adăuga noul eveniment, care va apărea apoi în lista **External name** din pagina **ActiveX Events** (vedeți figura 26.6).

Odată ce ați încheiat secvența de pași de mai sus, în cadrul clasei **CRotaryCtrl** din pagina **ClassView** ar trebui să fie afișată o nouă funcție, **FireRepositioned()**. Aceasta va genera evenimentul extern **Repositioned**, vizibil pentru aplicațiile care folosesc acest control ActiveX.

Acum trebuie să adăugați codul care va genera acest eveniment în momentul în care utilizatorul eliberează butonul stâng al mouse-ului (ceea ce arată că potențiometrul a fost deplasat). Înainte, însă, va trebui să stocați poziția curentă în cadrul clasei controlului. Pentru a adăuga o nouă variabilă membru în această clasă puteți să apelați caseta de

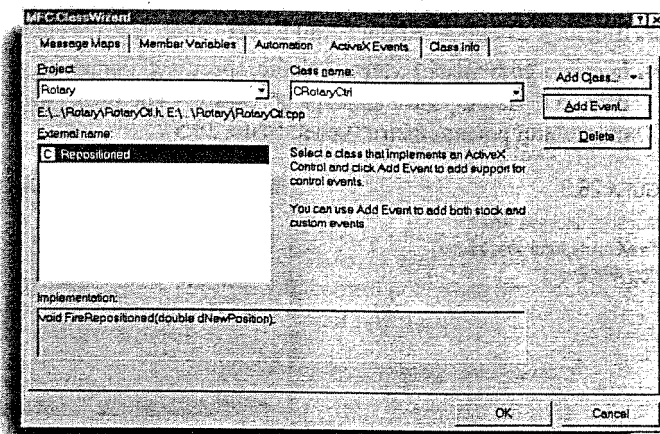


dialog New Member Variable sau puteţi să adăugaţi direct linia de mai jos în definiţia clasei CRotaryCtrl:

```
double m_dCurrentPosition;
```

FIGURA 26.6

Pagina **ActiveX Events** din ClassWizard afişează evenimentele generate curent de către control.



Această variabilă `double` va reţine un unghi cu valoarea între  $0^\circ$  şi  $360^\circ$ , reprezentând poziţia controlului potenţiometru începând de la Est ( $0^\circ$ ), trecând prin Nord ( $90^\circ$ ) şi Vest ( $180^\circ$ ), şi revenind la Est ( $360/0^\circ$ ). Valoarea variabilei poate fi stabilită pe baza unghiului calculat în funcţia `OnDraw()`, adăugând la sfârşitul acesteia linia următoare:

```
m_dCurrentPosition = dAngle * 57.2978 + 180.0;
```

În acest fel, unghiul calculat în radiani este transformat în grade şi rezultatul este înscris în noua variabilă membru `m_dCurrentPosition`.

Acum puteţi să generaţi în funcţia `OnLButtonUp()` evenimentul care va informa aplicaţia că utilizatorul a modificat poziţia controlului. În acest scop va trebui să adăugaţi un apel al noii funcţii care generează evenimentul (`FireRepositioned()`). Acesta poate fi adăugat după apelul `ReleaseCapture()` din funcţia de tratare `OnLButtonUp()`, transmiţând poziţia curentă a potenţiometrului, ca aici:

```
FireRepositioned(m_dCurrentPosition);
```

Acum, de fiecare dată când utilizatorul modifică poziţia potenţiometrului şi eliberează butonul mouse-ului, aplicaţia va primi din partea controlului un eveniment `Repositioned`. Acest eveniment poate fi tratat cu uşurinţă prin intermediul ClassWizard, aşa cum aţi văzut atunci când aţi folosit controale ActiveX în capitolul 9, „Utilizarea controalelor ActiveX”.

#### VEDEȚI...

- Pentru explicații privind mesaje și evenimentele din Windows, vedeți pagina 75.
- Pentru informații privind adăugarea unei funcții de tratare pentru un eveniment ActiveX, vedeți pagina 196.

## Crearea interfeței pentru proprietăți

Asemeni altor controale, controlul ActiveX pe care-l creați va dispune probabil de mai multe proprietăți pe care dumneavoastră sau un alt utilizator al controlului le puteți modifica la momentul proiectării – adică în timp ce controlul este folosit la proiectarea unei casete de dialog. Cel care utilizează controlul ar trebui, de asemenea, să aibă posibilitatea de a modifica aceste proprietăți prin codul aplicației.

### Cele trei tipuri de proprietăți

Proprietățile unui control ActiveX pot fi împărțite în trei categorii principale. Proprietățile ambientale sunt acele proprietăți moștenite de la containerul controlului cu scopul de a facilita o cât mai bună integrare a controlului în cadrul aplicației client. Proprietățile generice sunt comune tuturor controalelor, așa fiind culorile, fonturile și stările de activare. Proprietățile specifice sunt cele pe care le adăugați explicit în cadrul controlului, fiind specifice funcționării acestuia.

Proprietățile sunt asociate cu funcții de automatizare OLE, astfel încât controlul va putea fi utilizat și configurat de aplicații Visual C++, aplicații Visual Basic, un browser Web sau de orice alt program care știe să comunice prin intermediul interfețelor dispecer.

#### VEDEȚI...

- Pentru informații privind automatizarea OLE, vedeți pagina 589.

## Implementarea proprietăților generice

Atunci când adăugați proprietăți ale unui control, în realitate adăugați noi intrări în interfața dispecer OLE a controlului. Aceste intrări sunt asociate cu un nume unic, astfel încât și alte limbaje să poată localiza în interfața dispecer OLE a controlului proprietatea asociată cu numele respectiv. Există mai multe proprietăți generice comune tuturor controalelor ActiveX, așa cum se vede în tabelul 26.1. Două astfel de proprietăți sunt culoarea de afișare și culoarea de fundal, numele lor fiind `BackColor` și `ForeColor`. Proprietățile generice pot fi adăugate în cadrul unui control cu ajutorul ClassWizard. Urmăriți pașii de mai jos pentru a adăuga proprietatea generică `ForeColor`, care va permite aplicațiilor să modifice culoarea de afișare a controlului potenţiometru.

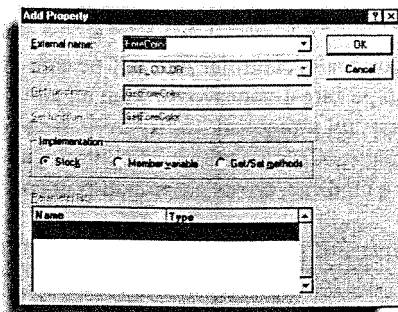
### Adăugarea unei proprietăți generice a unui control ActiveX cu ClassWizard

1. Apăsați **Ctrl+W** sau efectuați clic pe meniul **View** și selectați opțiunea **ClassWizard** pentru a apela ClassWizard.
2. Selectați pagina **Automation**.
3. Efectuați un clic pe butonul **Add Property** pentru a afișa caseta de dialog **Add Property**.
4. Selectați o proprietate generică din lista derulantă a casetei combinate **External Names**; pentru cazul de față selectați `ForeColor` (vedeți figura 26.7). După alegerea unei proprietăți generice, celelalte casete ar trebui să fie dezactivate.
5. Efectuați un clic pe **OK** pentru a adăuga proprietatea generică, iar aceasta va apărea apoi în lista **External Names** din pagina **Automation** a casetei de dialog ClassWizard.



FIGURA 26.7

Adăugarea unei proprietăți generice cu ajutorul casetei de dialog Add Property din ClassWizard.



TABELUL 26.1 Proprietățile generice disponibile pentru controalele ActiveX

| Numele proprietății | Accesată prin                               | Descriere                                |
|---------------------|---------------------------------------------|------------------------------------------|
| BackColorGet        | GetBackColor()<br>SetBackColor()            | Reține culoarea de fundal                |
| ForeColor           | GetForeColor()<br>SetForeColor()            | Reține culoarea de afișare               |
| Font                | GetFont()<br>SetFont()<br>InternalGetFont() | Reține fontul curent                     |
| Caption             | InternalGetText()                           | Reține conținutul etichetei              |
| Text                | InternalGetText()                           | La fel ca și Caption                     |
| BorderStyle         | m_sBorderStyle                              | ccNone sau ccFixedSingle (chenar simplu) |
| Appearance          | m_sAppearance                               | 1 pentru efect 3D, 0 pentru aspect plan  |

După ce adăugați proprietatea generică ForeColor, veți vedea că aceasta figurează în cadrul interfeței \_DRotary din pagina ClassView a secțiunii spațiului de lucru al proiectului. Puteți să modificați OnDraw() pentru a selecta la afișare o pensulă corespunzătoare cu această culoare de desenare, folosind în acest sens funcția GetForeColor():

```
CBrush brForeGnd(TranslateColor(GetForeColor()));
```

#### OLE\_COLOR și TranslateColor()

Folosiți funcția TranslateColor() pentru a transforma valorile de tip OLE\_COLOR în valori de tip COLORREF folosite de funcțiile standard din Windows. OLE\_COLOR extinde reprezentarea pe 24 de biți a culorilor COLORREF cu o serie de indicatori care specifică metode alternative de stocare a informațiilor de culoare. O astfel de alternativă este stocarea de indici pentru culorile standard din sistem, aceștia putând fi utilizați împreună cu funcția GetSysColor() pentru a determina culorile de sistem selectate pe o mașină locală. O altă alternativă este specificarea culorilor ca intrări într-o paletă, nu ca valori de culoare pe 24 de biți, ca în cazul COLORREF.

## Adăugarea unei pagini pentru proprietățile generice de culoare

Atunci când controlul este manipulat la momentul proiectării, puteți să permiteți utilizatorului controlului să selecteze mai ușor culoarea de afișare, adăugând o pagină pentru proprietățile generice de culoare ale controlului. Această pagină are identificatorul CLSID\_CColorPropPage, existând și o pagină pentru fonturi care are identificatorul CLSID\_CFontPropPage. În fișierul de implementare al clasei CRotaryCtrl, RotaryCtrl.cpp, puteți găsi o secțiune // Property Pages. Aici sunt enumerate paginile de proprietăți asociate controlului în mod curent. Atunci când adăugați controlul într-o casetă de dialog, în cadrul editorului de resurse, utilizatorul/proiectantul poate să efectueze un clic cu butonul drept pe suprafața controlului pentru a-i modifica proprietățile. Aceasta va duce la afișarea paginilor prezente în secțiunea menționată, alături de paginile General și All, așa cum am discutat în capitolul 9. La ora actuală, controlul potențiometrului are asociată o singură pagină de proprietăți, generată de ClassWizard:

```
BEGIN_PROPPAGEIDS(CRotaryCtrl, 1)
 PROPPAGEID(CRotaryPropPage::guid)
END_PROPPAGEIDS(CRotaryCtrl)
```

#### Modificarea numărului paginilor de proprietăți

Atunci când adăugați noi pagini de proprietăți, nu uitați să modificați și numărul total de pagini; în caz contrar, Visual Studio se va bloca la afișarea în cadrul editorului de resurse a proprietăților controlului.

Pagina corespunzătoare proprietăților generice de culoare se adăugă lesne, prin adăugarea intrării de mai jos după pagina implicită (CRotaryPropPage::guid) și actualizarea la doi a numărului de pagini specificat în macrodefiniția BEGIN\_PROPPAGEIDS:

```
BEGIN_PROPPAGEIDS(CRotaryCtrl, 2)
 PROPPAGEID(CRotaryPropPage::guid)
 PROPPAGEID(CLSID_CColorPropPage)
END_PROPPAGEIDS(CRotaryCtrl)
```

Odată efectuate aceste modificări, puteți să asamblați controlul și să-l testați prin adăugarea sa, cu ajutorul editorului de resurse, în cadrul casetei de dialog IDD\_ABOUTBOX\_ROTARY. Controlul va apărea complet negru, având culoarea de desenare stabilită la valoarea implicită (zero). Dacă apăsați Alt+Enter pentru a modifica proprietățile controlului, veți vedea că noua proprietate generică ForeColor apare în pagina All, având alăturată valoarea OLECOLOR corespunzătoare (vedeți figura 26.8). Prin efectuarea unui clic pe butonul ... puteți să selectați valoarea de culoare dorită în cadrul paginii Colors. Noua culoare va fi afișată automat, așa cum se vede în figura 26.9. După ce ați selectat o nouă culoare, închideți caseta de dialog cu proprietăți și veți vedea cum controlul este afișat cu noua culoare de desenare.

## Adăugarea proprietăților specifice

Puteți să adăugați proprietăți specifice controlului și să inserați într-o pagină de proprietăți controale care să permită accesarea acestor proprietăți. Spre exemplu, ați putea să afișați în

jurul potențiometrului gradată care să ajute utilizatorul să regleze poziția mai precis, permițând proiectantului casetei de dialog să specifice numărul de gradății afișate.

FIGURA 26.8

Pagina All din catalogul de proprietăți al controlului, conținând noua proprietate ForeColor.

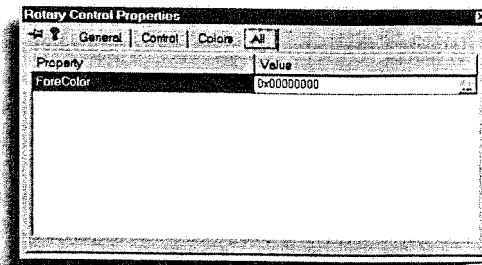
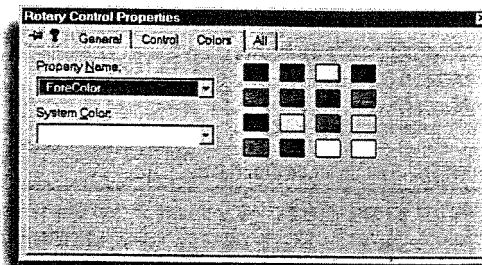


FIGURA 26.9

Pagina Colors cu proprietăți generice din catalogul de proprietăți al controlului.



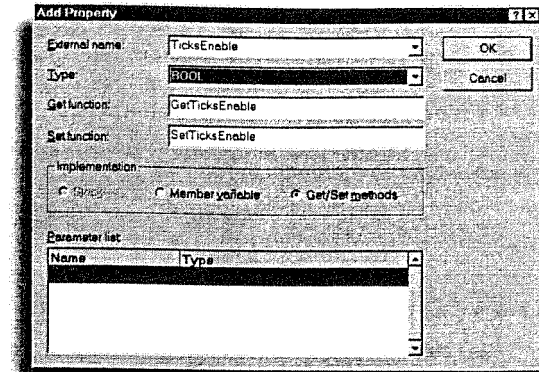
Prin intermediul paginii Automation din ClassWizard veți adăuga două proprietăți specifice, maniera fiind similară cu cazul proprietăților generice. În loc să alegeți o proprietate generică din caseta combinată External Names, puteți să introduceți numele proprietății specifice, așa cum ar fi TicksEnable, care va activa afișarea gradățiilor în jurul controlului. Pe măsură ce introduceți acest nume, în caseta Variable Name apare numele unei variabile m\_ticksEnable, iar în caseta Notification Function este generat automat numele unei funcții OnTicksEnableChanged. În loc să folosiți o variabilă membru, puteți să efectuați un clic pe butonul de opțiune Get/Set Methods pentru a utiliza metode Get și Set. În acest caz vor fi afișate numele funcțiilor corespunzătoare, GetTicksEnable pentru Get Function și SetTicksEnable pentru Set Function. Apoi selectați BOOL din caseta combinată Type ca tip al noii proprietăți, așa cum o ilustrează figura 26.10.

Efectuați un clic pe OK pentru a adăuga noile metode și inserați încă o proprietate, numită NumTicks, corespunzătoare numărului de gradății afișate, selectând și de această dată butonul de opțiune Get/Set Methods și specificând short în caseta Type a tipului proprietății.

În scopul implementării funcțiilor adăugate, selectați proprietatea dorită din lista External Name și efectuați un clic pe butonul Edit Code (din pagina Automation a casetei de dialog ClassWizard) pentru a putea edita funcțiile Get și Set corespunzătoare proprietății. Atunci când este apelată metoda Set, veți dori să stocați în cadrul clasei valoarea transmisă pentru noua proprietate, folosind-o apoi la afișare. Atunci când este apelată metoda Get, veți întoarce valoarea stocată a proprietății. Toate acestea sunt ilustrate în listingul 26.4.

FIGURA 26.10

Adăugarea unei proprietăți specifice prin intermediul casetei de dialog Add Property.



#### Numele externe ale proprietăților

Aceste nume externe ale proprietăților sunt similare cu numele externe folosite la automatizarea OLE. Ele sunt stocate în tabela unei interfețe IDispatch, astfel încât programele client (așa cum este containerul de test pentru controale ActiveX) să poată afla la momentul execuției proprietățile disponibile.

#### LISTINGUL 26.4 LST27\_4.CPP – Implementarea funcțiilor Get/Set pentru proprietățile specifice

```
1 BOOL CRotaryCtrl::GetTicksEnable()
2 {
3 // TODO: Add your property handler here
4 return m_bTicks;
5 }
6
7 void CRotaryCtrl::SetTicksEnable(BOOL bNewValue)
8 {
9 // TODO: Add your property handler here
10 m_bTicks = bNewValue;
11
12 SetModifiedFlag();
13 }
14
15 BOOL CRotaryCtrl::GetNumTicks()
16 {
17 // TODO: Add your property handler here
18
19 return m_sNumTicks;
20 }
21
```

(continuare)

## LISTINGUL 26.4 Continuare

```

22 void CRotaryCtrl::SetNumTicks(short nNewValue)
23 {
24 // TODO: Add your property handler here
25
26 m_sNumTicks = nNewValue;
27
28 SetModifiedFlag();
29 }

```

În listingul 26.4 sunt folosite două noi variabile membru ale clasei CRotaryCtrl (`m_bTicks` și `m_sNumTicks`) cu scopul de a stoca starea de activare și, respectiv, numărul de gradații. Metodele `Get/Set` pentru activarea afișării gradațiilor, între liniile 1 și 13, furnizează sau stabilesc starea de activare. Același lucru se întâmplă cu valoarea `m_sNumTicks` a proprietății `NumTicks`, între liniile 15 și 29. Aceste variabile membru trebuie adăugate în definiția clasei CRotaryCtrl, direct sau cu ajutorul casetei de dialog `Add Member Variable`:

```

short m_sNumTicks;
BOOL m_bTicks;

```

și este necesară inițializarea lor cu valori implicite în cadrul funcției constructor a clasei CRotaryCtrl, CRotaryCtrl::CRotaryCtrl(), ca aici:

```

// TODO: Initialize your control's instance data here.
m_bTicks = TRUE;
m_sNumTicks = 20;

```

Dacă expandați interfața `_Drotary` în cadrul paginii `ClassView`, veți vedea intrările corespunzătoare noilor proprietăți `TicksEnable` și `NumTicks`.

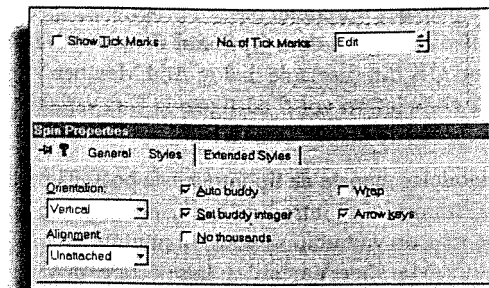
## Adăugarea într-o pagină de proprietăți a controalelor corespunzătoare proprietăților specifice

Noile metode `Get` și `Set` vor permite aplicațiilor care utilizează controlul să modifice proprietățile specifice, dar ar trebui să implementați și o interfață cu utilizatorul, care să permită modificarea acestor proprietăți la proiectarea dialogului, în cadrul editorului de resurse.

Selectați pagina `ResourceView` din secțiunea spațiului de lucru al proiectului și localizați caseta de dialog `IDD_PROPPAGE_ROTARY`. În această casetă de dialog pot fi adăugate controalele corespunzătoare noilor proprietăți specifice. Proprietatea `TicksEnable` poate fi reprezentată printr-o casetă de validare cu eticheta `Show Tick Marks` și identificatorul `IDC_TICKS_ENABLED`. Pentru reprezentarea numărului de gradații puteți să folosiți un control de editare numeric, având identificatorul `IDC_NUM_TICKS`. Alături de controlul de editare poate fi adăugat un control de modificare valorică, asociat corespunzător prin intermediul opțiunilor de stil (`Auto Buddy` și `Set Buddy Integer` din pagina `Styles` a casetei de dialog `Spin Properties`), permițând proiectantului să modifice cu ușurință numărul de gradații, așa cum se vede în figura 26.11.

FIGURA 26.11

Adăugarea de controale în pagina de proprietăți a controlului ActiveX.



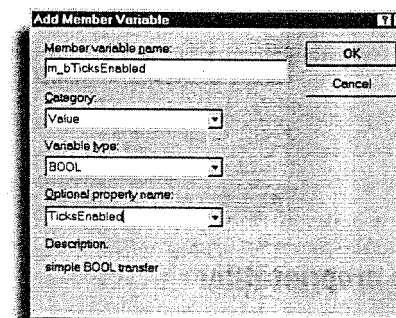
Maparea valorilor acestor controale peste proprietățile specifice ale controlului potențiometru se poate face lesne, apelând la `ClassWizard`. Pentru a mapa caseta de validare `Show Tick Marks` peste proprietatea `TicksEnable`, urmați secvența de pași „Maparea proprietăților specifice prin intermediul `ClassWizard`”.

### Maparea proprietăților specifice prin intermediul ClassWizard

1. Selectați controlul pe care doriți să-l mapați (casetă de validare `Show Tick Marks` din caseta de dialog `IDD_PROPPAGE_ROTARY`, pentru exemplul de potențiometru ActiveX) și apăsați `Ctrl+W` pe a apela `ClassWizard`, asigurându-vă că este selectată pagina `Member Variables`.
2. Selectați identificatorul `IDC_TICKS_ENABLED` din lista `Control IDs`.
3. Efectuați un clic pe butonul `Add Variable` pentru a mapa noua proprietate. Va fi afișată caseta de dialog `Add Member Variable`, ilustrată în figura 26.12.
4. Introduceți `m_bTicksEnabled` în caseta `Member Variable Name`.
5. Introduceți `TicksEnabled` în caseta `Optional Property Name`, așa cum se vede în figura 26.12.
6. Efectuați un clic pe `OK` pentru a adăuga noua mapare.

FIGURA 26.12

Maparea unei proprietăți specifice peste un control corespunzător, prin intermediul casetei de dialog `Add Member Variable`.



Puteți să repetați pașii anteriori pentru a mapa controlul `IDC_NUM_TICKS` (`Control ID`) peste variabila membru numită `m_sNumTicks`, având tipul `short` (`Variable Type`). Puteți

să efectuați și maparea variabilei peste o proprietate specifică (numită NumTicks), introducând NumTicks în caseta **Optional Property Name**. În momentul în care efectuați clic pe butonul **OK** din caseta de dialog Add Member Variable va fi creat codul necesar mapărilor, iar noile intrări vor fi adăugate în lista variabilelor mapate din pagina **Member Variables** a casetei de dialog ClassWizard.

În cazul variabilelor mapate de tip întreg este posibilă precizarea unui interval de validare, permițând utilizatorului să introducă exclusiv valori din acest interval (de exemplu, între 1 și 100). În acest scop va trebui să selectați noul întreg mapat (m\_sNumTicks) din lista afișată în pagina **Member Variables**; la selectare, în partea inferioară a paginii **Automation** vor apărea două controale de editare care vă permit să specificați intervalul de validare. Introduceți 1 în caseta **Minimum Value** și 100 în caseta **Maximum Value**, așa cum o arată figura 26.13, pentru a limita numărul de gradații între 1 și 100.

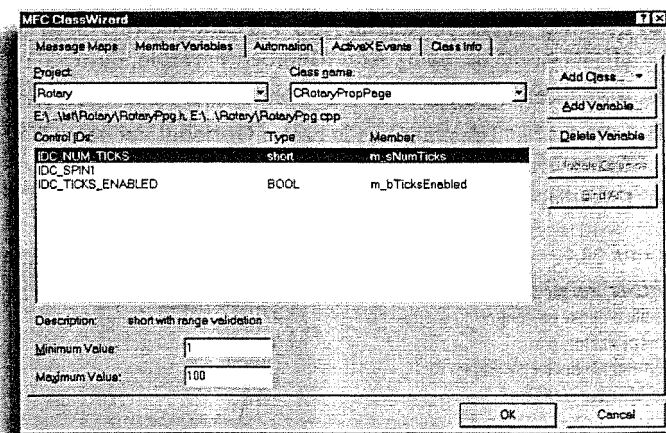
ClassWizard mapează aceste proprietăți din pagina de proprietăți în cadrul interfețelor OLE, adăugând în funcția DoDataExchange() a clasei CRotaryPropPage următoarele macrodefiniții speciale de tip DDP:

```
DDP_Check(pDX, IDC_TICKS_ENABLED, m_bTicksEnabled,
 _T("TicksEnable"));
DDP_Text(pDX, IDC_NUM_TICKS, m_sNumTicks, _T("NumTicks"));
```

Acum, de fiecare dată când proiectantul stabilește aceste opțiuni ale controlului ActiveX, actualizarea se va face prin intermediul metodelor Get și Set ale acestuia.

FIGURA 26.13

Stabilirea intervalului de validare pentru variabila membru m\_sNumTicks.



## Conservarea proprietăților

Atunci când modificați proprietățile unui control în cadrul editorului de resurse, aceste proprietăți își păstrează valoarea la folosirea sau configurarea ulterioară a controlului. Aceste proprietăți sunt serializate într-o formă *permanentă* de stocare pentru fiecare

instanță a controlului, astfel încât proprietățile stabilite la adăugarea unui control într-o casetă de dialog își conservă valoarea până la următoarea editare a casetei de dialog.

În acest scop, controlul dispune de o funcție `DoPropExchange` care este apelată ori de câte ori este necesară încărcarea sau salvarea proprietăților controlului. Pentru serializarea proprietăților specifice puteți să adăugați aici macrodefiniții `PX_` de actualizare a proprietăților, așa cum o ilustrează listingul 26.5.

### LISTINGUL 26.5 LST27\_5.CPP – Conservarea proprietăților specifice prin adăugarea de macrodefiniții `PX` în cadrul funcției `DoPropExchange`

```
1 ///
2 // CRotaryCtrl::DoPropExchange - Persistence support
3
4 void CRotaryCtrl::DoPropExchange(CPropExchange* pPX)
5 {
6 ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
7 COleControl::DoPropExchange(pPX);
8
9 // TODO: Call PX_ function for each persistent cust
10
11 // ** Serializam optiunea de afisare a gradatiilor
12 PX_Bool(pPX, _T("TicksEnable"), m_bTicks, TRUE); ❶
13
14 // ** Serializam optiunea privind numarul de gradatii
15 PX_Short(pPX, _T("NumTicks"), m_sNumTicks, 20);
16 }
```

❶ Macrodefinițiile `PX_` sunt folosite pentru a realiza asocierea între interfața unei proprietăți și o variabilă membru a clasei controlului.

Linia 12 are ca efect conservarea proprietății `m_bTicks` prin intermediul macrodefiniției `PX_Bool`, iar proprietatea `m_sNumTicks` este conservată prin intermediul macrodefiniției `PX_Short` din linia 15. Primul parametru transmis acestor macrodefiniții este un pointer `pPX` la un obiect `CPropExchange`. Al doilea parametru este numele proprietății OLE. Cel de-al treilea parametru reprezintă valoarea locală a proprietății, specificată ca variabilă membru. Ultimul parametru precizează o valoare implicită pentru inițializare. Există macrodefiniții `PX_` pentru aproape toate tipurile de date care v-ar putea interesa, printre care `PX_Double`, `PX_Color` și `PX_String`.

Observați, de asemenea, că apelul funcției `ExchangeVersion()` din linia 6 se ocupă de stocarea numerelor de versiune. Acestea pot fi testate în eventualitatea în care adăugați proprietăți suplimentare într-o versiune ulterioară a controlului, după livrarea sa. Astfel, veți actualiza doar proprietățile corespunzătoare versiunii respective, păstrând compatibilitatea cu versiuni vechi ale proprietăților atunci când utilizatorul modernizează controlul la o versiune mai nouă.

Acum puteți să adăugați la sfârșitul funcției `OnDraw()` codul necesar pentru afișarea gradațiilor, ca în listingul 26.6.

#### LISTINGUL 26.6 LST27\_6.CPP – Adăugarea la sfârșitul funcției `OnDraw()` a clasei `CRotary` a codului necesar pentru afișarea gradațiilor

```
1 // Verificam daca este activata afisarea gradatiilor
2 if (m_bTicks)
3 {
4 // Iteram in radiani de la -2*PI la +2*PI
5 const double dPi = 3.14185;
6 double r = -2.0 * dPi;
7 for(int i=0; i<m_sNumTicks; i++)
8 {
9 // Ne plasam intr-un punct in exteriorul corpului circular
10 nXPos = ptMid.x + (int)(cos(r) * dRadX * 1.05);
11 nYPos = ptMid.y + (int)(sin(r) * dRadY * 1.05);
12 pdc->MoveTo(CPoint(nXPos, nYPos));
13
14 // Trasam o linie si mai in exterior, pe post de gradatie
15 nXPos = ptMid.x + (int)(cos(r) * dRadX * 1.15);
16 nYPos = ptMid.y + (int)(sin(r) * dRadY * 1.15);
17 pdc->LineTo(CPoint(nXPos, nYPos)); 1
18
19 // Incrementam unghiul
20 r += dPi / (m_sNumTicks / 2.0);
21 }
22 }
```

1 Gradațiile sunt desenate radial, în exteriorul corpului circular al potențiometrului.

În listingul de mai sus, în linia 2 se testează membrul `m_bTicks` pentru a verifica dacă este activată afișarea gradațiilor. Bucla din linia 7 execută liniile dintre 9 și 20 de un număr de ori egal cu numărul de gradații înscris în `m_sNumTicks`. Liniile de la 10 la 12 deplasează cursorul curent într-o poziție care depinde de unghiul `r`, acest unghi fiind incrementat în linia 20 de la  $-2\text{PI}$  la  $2\text{PI}$ . Liniile de la 14 la 17 trasează din acest punct un segment radial înspre exterior, astfel fiind afișate gradații de jur împrejurul corpului circular al potențiometrului.

#### VEDEȚI...

- Pentru amănunte privind trasarea liniilor și a cercurilor, vedeți pagina 349
- Pentru informații despre automatizarea OLE, vedeți pagina 589

## Compilarea și înregistrarea controlului

Fișierele folosite și generate la compilarea unui control ActiveX diferă puțin de cele din cazul aplicațiilor obișnuite, de tip dialog sau SDI/MDI, acest fapt datorându-se interfețelor COM/OLE pe care le expune controlul. Codul controlului este generat într-un fișier `.ocx`,

care este de fapt un `.dll` cu alt nume. În consecință, codul sursă include funcțiile de inițializare și înregistrare tipice pentru `.dll`, precum și fișierele de definiție. Este posibilă integrarea într-un singur fișier `.dll` a mai multor controale, înlesnind astfel distribuția.

Deși etapele asamblării se desfășoară în mod automat, este bine să știți ce fișiere se folosesc și ce fișiere sunt create și trebuie furnizate la distribuția controlului.

## Diferitele fișiere sursă

Dacă selectați pagina `FileView` din secțiunea spațiului de lucru al proiectului `Rotary`, de tip control ActiveX, în catalogul `Source Files` puteți vedea următoarele fișiere:

- *Rotary.cpp*. Aici se află numerele de versiune și identificatorul global unic al controlului. Sunt implementate, de asemenea, funcțiile `DllRegisterServer()`, `DllUnregisterServer()`, `InitInstance()` și `ExitInstance()`, necesare pentru rularea și înregistrarea controlului.
- *Rotary.def*. Aici se află definițiile funcțiilor exportate de către DLL.
- *Rotary.odl*. Acest fișier este scris în limbajul de definiție a obiectelor și conține definițiile interfețelor COM/OLE folosite de control, generând fișierul bibliotecă de tipuri care permite diverselor aplicații și limbaje să utilizeze controlul.
- *Rotary.rc*. Fișierul de resurse care conține paginile de proprietăți specifice și alte resurse ale proiectului.

### Înregistrarea controlului la momentul compilării

Operația de înregistrare, care folosește programul `regsvr32.exe`, actualizează registrul sistemului pentru a înregistra diferenți identificatori de clasă necesari noului control. Această operație este inclusă în procesul de compilare pentru comoditate, deoarece este probabil că veți dori să rulați controlul pe propriul calculator pentru a-l testa. Trebuie să rețineți că operația de înregistrare trebuie efectuată și pe calculatoarele utilizatorilor finali ai controlului, fiind de regulă inclusă în scriptul de instalare/configurare.

- *RotaryCtl.cpp*. Fișierul sursă C++ care conține codul de implementare a controlului.
- *RotaryPpg.cpp*. Fișierul sursă C++ care conține codul de implementare pentru paginile de proprietăți ale controlului.

Prima etapă a compilării folosește compilatorul MIDL (Microsoft Interface Definition Language – limbajul de definiție a interfețelor) pentru a prelucra fișierul `.odl`. Așa cum am menționat anterior, aceasta va duce la generarea definițiilor de interfețe și a bibliotecii de tipuri asociată. În continuare, fișierele sursă sunt compilate în mod obișnuit. În fine, este executat programul `regsvr32.exe` cu scopul de a înscrie în registrul sistemului intrările corespunzătoare pentru control și interfețele sale.

## Crearea fișierelor pentru biblioteca de tipuri și pentru licențiere

Odată încheiată compilarea, în directorul de ieșire `Rotary/Debug` puteți găsi un fișier `Rotary.tlb`. Acesta este generat de compilator și va fi utilizat la generarea claselor dispecer sau interfață atunci când controlul este încorporat într-un proiect de aplicație. La distribuirea

controlului, pe lângă fișierul .ocx care conține codul ar trebui să furnizați și acest fișier .tlb. Ați putea dori, de asemenea, să distribuiți fișierele de definiție .lib și .exp pentru DLL, precum și fișierul .lic generat în cazul în care ați selectat suportul pentru licențiere.

#### Licențierea controalelor ActiveX

Atunci când un proiectant licențiat adaugă controlul dumneavoastră în propria aplicație, este generată o cheie de licență unică, încapsulată în control și stocată în programul aplicației. Pentru generarea acestei chei și a informațiilor asociate privind drepturile de autor, proiectantul licențiat trebuie să aibă fișierul .lic adecvat. Astfel, va putea să distribuie utilizatorilor finali propria aplicație care folosește controlul dumneavoastră. În momentul în care utilizatorul rulează această aplicație, containerul controlului trebuie să apeleze funcția `CreateInstanceLic()` pentru a crea o instanță de licență. Aplicația transmite propria cheie de licență, care este comparată cu cheia de licență încapsulată în control, aplicația continuându-și execuția în cazul în care cele două chei sunt identice.

#### VEDEȚI...

➤ Pentru informații privind bibliotecile de tipuri, vedeți pagina 594.

### Înregistrarea controlului

În momentul în care distribuiți controlul utilizatorilor finali, singurul fișier care trebuie furnizat este fișierul .ocx, alăturând eventual programul `regsvr32.exe`, care poate fi distribuit liber cu acceptul Microsoft. Dar controlul trebuie înregistrat în sistemele utilizatorilor, rulând `regsvr32.exe` ca aici (efectul este înregistrarea controlului potențiometru):

```
(regsvr32.exe /s Rotary.ocx
```

Opțiunea `/s` determină desfășurarea operației în mod acuns; în caz contrar, va fi afișată o casetă de mesaj care confirmă înregistrarea cu succes a controlului.

### Testarea cu ajutorul containerului de test pentru controale ActiveX

Dacă adăugarea controlului în caseta de dialog About reprezintă o modalitate brută, dar simplă de testare a controlului, o soluție mult mai bună este folosirea containerului de test pentru controale ActiveX, oferit împreună cu compilatorul. Acest program este destul de sofisticat și vă permite să stabiliți toate proprietățile asociate controlului, să jurnalizați evenimentele generate de către control și să testați interfața cu utilizatorul a proprietăților.

Containerul de test pentru controale ActiveX poate fi apelat efectuând un clic pe meniul **Tools** și selectând opțiunea **ActiveX Control Test Container**.

### Selectarea și inserarea controlului

Inserarea noului control ActiveX în cadrul containerului de test se face efectuând un clic pe meniul **Edit** și selectând opțiunea **Insert OLE Control** sau efectuând clic pe primul buton de pe bara cu instrumente. Va fi afișată caseta de dialog **Insert OLE Control**, din care puteți selecta controlul dorit. Selectați din listă controlul **Rotary** (în partea inferioară) și efectuați un clic pe **OK** pentru a afișa acest control.

#### Decuparea la desenarea în cadrul controlului

Containerul de test pentru controale ActiveX nu efectuează operații de decupare așa cum este cazul unui container adevărat dintr-o aplicație. Astfel aveți posibilitatea de a identifica un comportament inadecvat al controlului în sensul desenării peste granițele containerului. Este bine să încercați să rezolvați problemele de decupare care apar în containerul de test, mai ales în cazul în care la creare ați selectat opțiunea **Unclipped Device Context**.

Potențiometrul va apărea complet negru (valoarea implicită pentru `ForeColor` este zero). Puteți să redimensionați controlul și să-l plasați oriunde pe suprafața containerului.

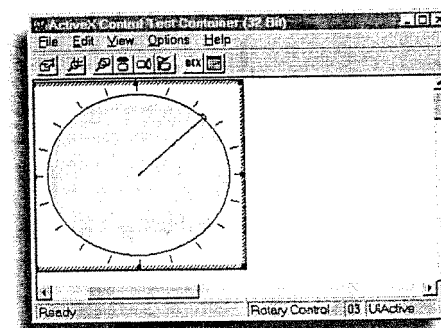
### Testarea proprietăților controlului

Dacă deschideți meniul **Edit** după inserarea controlului, veți vedea că în partea sa inferioară a apărut o nouă opțiune **Properties...Rotary Control Object**. Selectați această opțiune pentru a afișa paginile de proprietăți ale controlului. Ar trebui să vedeți pagina cu proprietăți specifice, care permite activarea sau dezactivarea gradațiilor și specificarea numărului de gradații. Acestea și proprietatea `ForeColor` pot fi modificate pentru a testa diferitele efecte, așa cum o ilustrează figura 26.14.

De asemenea, ar trebui să puteți interacționa cu controlul, efectuând un clic pe suprafața sa și trăgând indicatorul de poziție pentru a modifica poziția potențiometrului.

FIGURA 26.14

Testarea proprietăților controlului Rotary în cadrul containerului de test.



### Testarea proprietăților ambientale

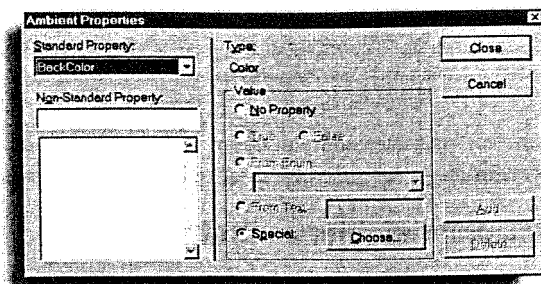
Este posibil, totodată, să stabiliți proprietățile ambientale ale containerului, efectuând clic pe meniul **Edit** și selectând opțiunea **Set Ambient Properties**. Aceasta va afișa caseta de dialog **Ambient Properties**, înfățișată în figura 26.15. Proprietatea ambientală pe care doriți să o modificați se selectează din caseta combinată **Standard Property**. Apoi se stabilește valoarea dorită, cu ajutorul controalelor aflate în partea dreaptă. Dacă selectați proprietatea ambientală `BackColor`, de care ține cont controlul potențiometru, și apoi efectuați un clic pe butonul **Choose**, va fi afișată o listă de culori din care puteți alege noua culoare ambientală de fundal. După ce selectați această culoare, efectuați un clic pe butonul **Close** pentru a închide caseta de dialog. Controlul ar trebui să fie afișat acum cu noua culoare de fundal. (Pentru a vedea efectul, s-ar putea



să fie necesară suprapunerea unei alte ferestre peste containerul de test, pentru ca apoi să descoperiți controlul și să forțați astfel redesenarea sa.)

FIGURA 26.15

Stabilirea proprietăților ambientale ale containerului de test pentru controale ActiveX.

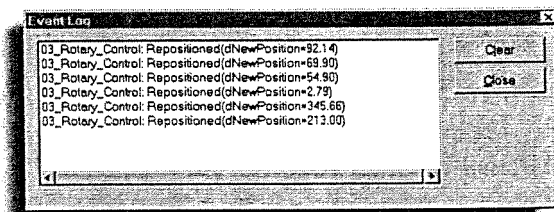


### Jurnalizarea evenimentelor generate

Dacă vreți să vă asigurați că evenimentele sunt generate în mod corespunzător, efectuați un clic pe meniul **View** și selectați opțiunea **Event Log**, afișând astfel caseta de dialog **Event Log**. Acum puteți să modificați poziția potențiometrului, iar la eliberarea butonului mouse-ului unghiul noii poziții va apărea în caseta de dialog **Event Log**, odată cu generarea evenimentului **Repositioned** (vedeți figura 26.16).

FIGURA 26.16

Jurnalizarea în fereastra **Event Log** a evenimentelor generate de control.



## CAPITOLUL

# 27

## Utilizarea depanatorului integrat

Asamblarea programelor cu informații pentru depanare sau optimizate pentru versiunea finală

Identificarea erorilor prin execuția pas cu pas a codului

Urmărirea ferestrelor și a mesajelor corespunzătoare



## Crearea informațiilor pentru depanare și inspectare

O mare parte din dezvoltarea aplicației o reprezintă *depanarea* programului. Crearea de software implică, în strânsă legătură, proiectarea, implementarea și depanarea aplicațiilor.

Visual C++ oferă un mediu de depanare complex și o serie de instrumente pentru depanare care ajută substanțial la dezvoltarea programelor. Puteți să identificați rapid problemele existente, să inspectați valorile variabilelor și să urmăriți fluxul de execuție al programului prin propriul cod și prin codul MFC.

Instrumente precum Spy++ vă pot informa asupra mesajelor transmise între Windows și propria aplicație, permițându-vă să spionați aplicațiile pentru a vedea ce controale conține interfața cu utilizatorul și care sunt tipurile și stilurile acestora.

## Utilizarea configurațiilor pentru depanare și pentru versiunea finală

La asamblarea aplicației puteți să alegeți între două configurații principale ale compilatorului: Debug (pentru depanare) și Release (pentru versiunea finală). Aceste configurații pot fi selectate efectuând un clic pe meniul **Project** și selectând opțiunea **Settings** sau apăsând Alt+F7, ceea ce va duce la afișarea casetei de dialog Project Settings (vedeți figura 27.1). Configurațiile principale se află în partea superioară și pot fi selectate în cadrul casetei combinate. La selectarea unei configurații, modificările efectuate asupra opțiunilor din paginile afișate în partea dreaptă sunt reținute în configurația respectivă. Asamblarea aplicației se va realiza pe baza opțiunilor din configurația curentă, fiind totodată posibil să selectați opțiunea **All Configurations** pentru a asambla și modifica simultan toate configurațiile.

### Stabilirea configurației curente a proiectului

Configurația curentă a proiectului constă dintr-o mulțime de instrucțiuni pe care compilatorul le va lua în considerare la crearea programului executabil. După dezvoltarea aplicației într-o configurație pentru depanare, veți dori să utilizați configurația pentru versiunea finală cu scopul de a produce un program executabil mic și rapid pe care să-l oferiți utilizatorilor.

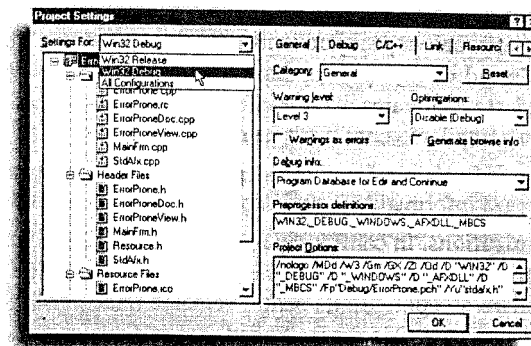
Selectarea unei configurații se face efectuând un clic pe meniul **Build** și selectând opțiunea **Set Active Configuration**. În consecință, va fi afișată caseta de dialog Set Active Project Configuration, conținând lista configurațiilor de proiect disponibile. Puteți, așadar, să efectuați un dublu clic pe configurația Release pentru a determina compilatorul să o utilizeze în locul configurației Debug.

Ambele configurații pentru depanare și pentru versiunea finală, sunt generate la crearea unui nou proiect; codul obiect produs în fiecare dintre cazuri este, însă, foarte diferit. În configurația pentru depanare, procesul de asamblare va avea ca rezultat un program executabil mare și destul de lent. Aceasta se întâmplă deoarece sunt incluse o mulțime de informații pentru depanare, fiind dezactivate orice optimizări la compilare.

La compilarea aceleiași aplicații în configurația pentru versiunea finală, rezultatul va fi un program executabil mic și rapid, însă nu veți mai putea să executați codul sursă pas cu pas sau să primiți mesaje de depanare.

FIGURA 27.1

Pagina C/C++ din caseta de dialog Project Settings.



În mod normal, pe parcursul dezvoltării unei aplicații veți folosi configurația pentru depanare a compilatorului, astfel încât să puteți identifica și depana cu ușurință problemele existente în cod. Odată ce ați terminat aplicația și sunteți gata să o distribuiți, puteți să selectați configurația pentru versiunea finală și să generați un program mic și rapid pentru beneficiul utilizatorilor.

### Testarea versiunii finale

Înainte de a oferi utilizatorilor aplicația asamblată în configurația pentru versiunea finală, aveți grijă întotdeauna să o testați în întregime. Este posibil să apară erori din cauza inserării de cod în macrodefinițiile **ASSERT** (despre care vom discuta mai târziu în acest capitol), care acum sunt înlăturate, sau ca efect al unor optimizări de viteză și memorie.

### VEDEȚI...

► Pentru mai multe informații despre configurarea și administrarea proiectelor, vedeți pagina 51.

## Stabilirea opțiunilor și a nivelelor pentru depanare

În pagina C/C++ a casetei de dialog Project Settings pot fi stabilite o multitudine de opțiuni și nivele pentru depanare. Această pagină este accesibilă din cadrul meniului **Project**, selectând opțiunea **Settings** (sau apăsând Alt+F7) și apoi selectând eticheta de pagină C/C++.

Având selectată opțiunea **General** în caseta **Category**, iată opțiunile disponibile:

- **Warning Level.** Acesta este nivelul la care sunt generate mesajele de avertizare pe parcursul compilării. Poate fi selectat oricare dintre nivelele prezentate în tabelul 27.1. Nivelul implicit este **Level 3**, care este destul de sensibil, deși mulți programatori buni de C++ preferă să aleagă **Level 4** pentru a fi avertizați cât mai bine asupra potențialelor probleme. **Level 1** și dezactivarea avertismentelor (**None**) nu trebuie folosite decât în situații speciale, deoarece acestea avertizează doar asupra problemelor grave (sau deloc).
- **Warnings as Errors.** Dacă selectați această opțiune, mesajele de avertizare sunt afișate ca erori și, prin urmare, întrerup procesul de asamblare.

- **Generate Browse Info.** Dacă selectați această opțiune, compilatorul va genera informații care vă pot ajuta la localizarea funcțiilor, a simbolurilor și a relațiilor dintre clase în cadrul unei ferestre Browse (despre care vom discuta în secțiunea următoare). Din păcate, generarea acestor informații utile poate mări destul de mult timpul necesar pentru compilarea unor proiecte mari (cele în care aveți cel mai mult nevoie).
- **Debug Info.** Această opțiune vă permite să specificați nivelul la care compilatorul generează informațiile pentru depanare, așa cum o arată tabelul 27.2.
- **Optimizations.** În configurația pentru depanare veți dezactiva de regulă orice optimizări, deoarece acestea interferează cu procesul de depanare și măresc timpul de compilare. În configurația pentru versiune finală, însă, puteți decide să maximizați viteza aplicației (**Maximize Speed**) sau să-i minimizați dimensiunea (**Minimize Size**), fiind posibil și un compromis care încearcă satisfacerea ambelor deziderate.
- **Preprocessor Definitions.** Aici sunt specificate definițiile folosite la compilarea programului. Aceste definiții pot fi utilizate împreună cu comenzile de preprocesare `#ifdef`, `#else` și `#endif`, în scopul compilării anumitor secțiuni de cod în funcție de configurație. `_DEBUG` este definit implicit în cadrul configurației pentru depanare și poate fi folosit pentru a determina compilarea unor secvențe de cod exclusiv în această configurație, ca în exemplul următor:

```
int a = b * c / d + e
#ifdef _DEBUG
CString strMessage;
strMessage.Format("Result of sum was %d", a);
AfxMessageBox(strMessage);
#endif
```

În acest caz, codul care afișează caseta de mesaj va fi compilat și executat doar atunci când aplicația este compilată în configurația pentru depanare. În momentul în care selectați configurația pentru versiunea finală, acest cod nu va mai fi compilat în cadrul executabilului.

- **Project Options.** Compilatorul propriu-zis rulează ca o aplicație de tip consolă, opțiunile exprimate în Developer Studio fiind transformate în parametri transmiși prin linia de comandă. În această casetă de editare puteți să adăugați parametri adiționali corespunzători unor opțiuni mai obscure, pentru care nu există o interfață cu utilizatorul.

TABELUL 27.1 Nivelele de avertizare în cadrul compilatorului

| Nivel   | Avertismente generate                            |
|---------|--------------------------------------------------|
| None    | Nici unul                                        |
| Level 1 | Doar cele mai grave                              |
| Level 2 | Mesaje mai puțin grave                           |
| Level 3 | Nivelul implicit (avertismente rezonabile)       |
| Level 4 | Sensibilitate maximă (bun pentru perfecționiști) |

TABELUL 27.2 Opțiuni pentru informațiile de depanare

| Opțiune                                | Informații de depanare generate                                                                                                                                                                                          |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| None                                   | Nu sunt generate informații de depanare – opțiune folosită de regulă în configurațiile pentru versiune finală.                                                                                                           |
| Line Numbers Only                      | Sunt generate doar numere de linii, care permit localizarea în codul sursă a funcțiilor și variabilelor globale. În schimb, timpul de compilare și dimensiunea executabilului sunt reduse.                               |
| C 7.0-Compatible                       | Sunt generate informații de depanare compatibile cu Microsoft C 7.0. Toate aceste informații sunt plasate în fișierele executabile, crescându-le dimensiunea, dar permițând în totalitate depanarea simbolică.           |
| Program Database                       | Această opțiune duce la crearea unui fișier cu extensia .pdb care conține un maxim de informații de depanare, mai puțin informații pentru editare și continuare.                                                         |
| Program Database for Edit and Continue | Aceasta este opțiunea implicită și tipică pentru depanare. Este creat un fișier .pdb care conține un maxim de informații de depanare, generând și informațiile necesare pentru noua facilități de editare și continuare. |

#### Efectul asupra parametrilor de compilare

Mulțimea de opțiuni exprimate este tradusă într-o lungă listă de parametri, transmisă compilatorului prin linia de comandă. Spre exemplu, opțiunile privind informațiile de depanare sunt traduse în parametri `/Z` pentru linia de comandă. Aceștia sunt `/Zd` pentru generarea numerelor de linii, `/Z7` pentru compatibilitatea cu C7, `/Zi` pentru baza de date a programului și `/Z1` pentru facilitățile de editare și continuare. Modificarea opțiunilor pentru informațiile de depanare se reflectă corespunzător în caseta Project Options.

## Crearea și utilizarea informațiilor pentru inspectare

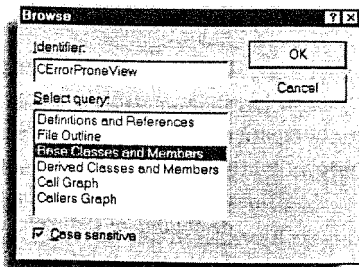
Pentru inspectarea detaliată a codului sursă puteți folosi instrumentul numit **Source Browser** (inspector de cod). Acest instrument poate deveni neprețuit în cazul în care examinați codul scris de altcineva sau dacă reveniți asupra propriului cod cu care nu ați mai avut contact o vreme.

Pentru a putea utiliza inspectorul de cod, la compilarea aplicației trebuie să fie selectată opțiunea **Generate Browse Info**, aflată în pagina C/C++ a casetei de dialog **Project Settings**. Apelarea acestui instrument se face apăsând **Alt+F12** sau efectuând un clic pe meniul **Tools** și selectând opțiunea **Source Browser** (la prima apelare vi se va solicita compilarea informațiilor pentru inspectare).

Caseta de dialog afișată inițial de inspectorul de cod solicită un identificator (**Identifier**) care să fie supus inspecției (așa cum se vede în figura 27.2). Acest identificator poate să fie un nume de clasă, un nume de structură, un nume de funcție sau numele unei variabile globale sau locale din cadrul aplicației. Odată introdus un identificator, butonul **OK** este activat și puteți să inspectați detaliile privind identificatorul respectiv.

FIGURA 27.2

Caseta de dialog Browse solicită un simbol care să fie supus inspecției.

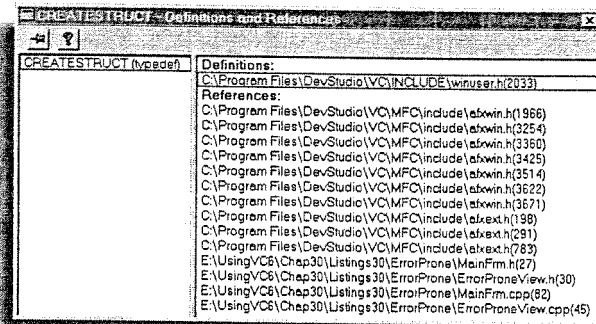


Lista **Select Query** oferă mai multe opțiuni privind detaliile inspectate în legătură cu simbolul specificat. Puteți alege oricare dintre opțiunile următoare:

- **Definitions and References.** Această opțiune afișează toate fișierele în care apare identificatorul specificat, precizând dacă este vorba despre referințe la identificator (locuri în care acesta este folosit în cod) sau definiții ale acestuia (locuri în care identificatorul este definit), așa cum o ilustrează figura 27.3. Pentru fiecare apariție este afișat numele fișierului și numărul liniei de cod. Dacă efectuați un dublu clic pe o referință sau o definiție, codul respectiv va fi încărcat și afișat la linia în cauză în cadrul editorului din Developer Studio. Această opțiune este foarte utilă pentru a urmări toate locurile în care este folosită o anumită variabilă sau funcție.

FIGURA 27.3

Inspectorul de cod afișează definiții și referințe.



- **File Outline.** Această opțiune afișează toate clasele, datele, funcțiile, macrodefinițiile și tipurile definite în fișierul specificat (ca identificator), după cum o arată figura 27.4. Puteți să filtrați fiecare tip de informație apăsând butoanele corespunzătoare din partea superioară a ferestrei inspectorului.
- **Base Classes and Members.** Aceasta este incontestabil una dintre cele mai utile opțiuni oferite de inspectorul de cod. Prin specificarea unei clase ca identificator, este afișată întreaga ierarhie de clase, cu funcțiile și variabilele membru de la fiecare nivel (vedeți figura 27.5). Și aici puteți stabili opțiuni de filtrare pentru a afișa doar anumite tipuri de funcții și variabile membru.

FIGURA 27.4

Sumarul unui fișier afișat de către inspectorul de cod.

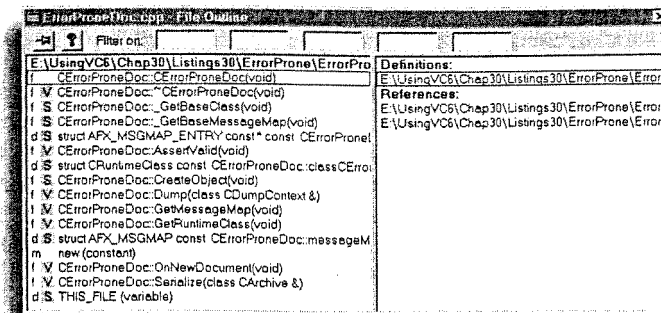


FIGURA 27.5

Afișarea în cadrul inspectorului de cod a claselor de bază și a membrilor acestora.

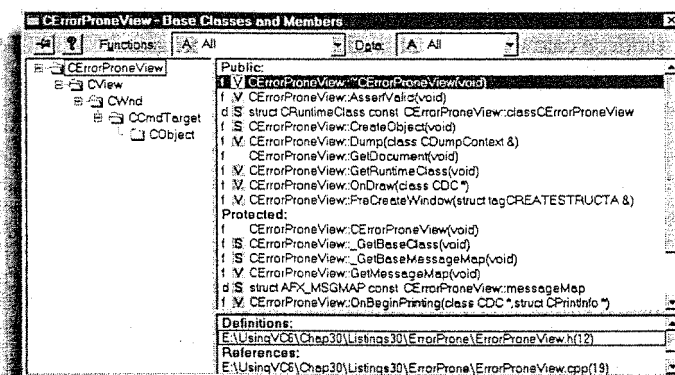
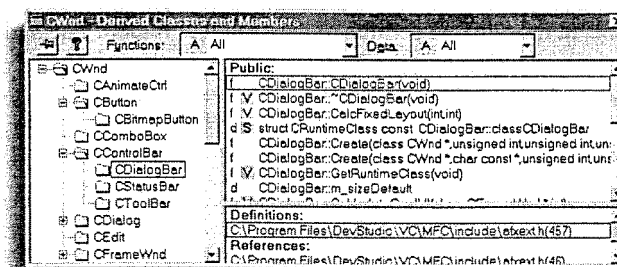


FIGURA 27.6

Afișarea în cadrul inspectorului de cod a claselor derivate din CWnd și a membrilor acestora.



- **Call Graph.** Această opțiune afișează toate funcțiile care sunt apelate de un anumit identificator, împreună cu fișierele în care acestea sunt definite și implementate. Astfel aveți posibilitatea de a urmări un potențial flux de execuție al programului.
- **Callers Graph.** Această opțiune este complementară celei anterioare, afișând toate funcțiile care apelează identificatorul specificat. Astfel aveți posibilitatea de a determina posibili apelanți ai funcției specificate.

## Utilizarea depanării la distanță și a depanării imediate

Depanatorul conține instrumente care vă permit să depanați un program ce rulează pe un alt calculator (chiar și în Internet, prin TCP/IP). Această facilitare poate fi utilă în cazul în care doriți să testați o aplicație în alte medii decât calculatorul pe care s-a desfășurat procesul de dezvoltare. În acest scop, în cele două sisteme trebuie să se afle exact aceleași versiuni de fișiere .dll și .exe. După încărcarea proiectului, depanarea sa se face prin intermediul unui director partajat de pe calculatorul aflat la distanță, înscriind calea și numele fișierului .exe local în caseta de editare **Executable for debug session** (aflată în pagina **Debug** din caseta de editare Project Settings).

Numele complet al fișierului .exe trebuie introdus și în caseta de editare **Remote Executable Path and File Name** din partea inferioară a paginii **Debug**, lăsând caseta **Working Directory** goală. În continuare, puteți să lansați monitorul pentru depanare la distanță pe calculatorul aflat la distanță, rulând programul MSVCMON.EXE și conectându-vă la el prin efectuarea unui clic pe meniul **Build** și selectarea opțiunii **Debugger Remote Connection**.

În cadrul casetei de dialog Remote Connection puteți să selectați **Local**, pentru o depanare prin intermediul unui director partajat, sau **Remote**, pentru depanarea prin conexiune TCP/IP. (Adresa poate fi specificată efectuând un clic pe **Settings**.) În acest fel veți stabili legătura cu monitorul aflat la distanță care va declanșa sesiunea de depanare la distanță.

### Instalarea fișierelor necesare pentru depanarea la distanță

Pentru rularea monitorului pentru depanare la distanță pe un alt calculator sunt necesare următoarele fișiere: MSVCMON.EXE, MSVCRT.DLL, TLN0T.DLL, DM.DLL, MSVCP50.DLL și MSDIS100.DLL. Aceste fișiere se găsesc în subdirectorul ..\Microsoft Visual Studio\Common\MSDev98\bin.

Depanarea imediată vă permite să depanați un program care a fost rulat normal (nu prin intermediul depanatorului) și apoi a produs o problemă. Dacă aveți Visual C++ instalat pe un calculator și această facilitare este activată, orice program care comite o eroare este încărcat într-o nouă instanță de Developer Studio, pregătit pentru depanare și afișând codul care a provocat blocarea.

Aceasta produce de multe ori zămbete atunci când se blochează chiar Developer Studio și este lansată o altă instanță a sa, afișând codul în limbaj de asamblare în care a survenit eroarea și permițându-vă să-l depanați. Depanarea imediată poate fi foarte utilă pentru depanarea propriilor aplicații atunci când acestea se blochează neașteptat (de regulă, într-o demonstrație în fața șefului). Pentru activarea acestei opțiuni, efectuați un clic pe meniul **Tools** și selectați **Options**, deschizând astfel caseta de dialog Options. Apoi accesați pagina **Debug** și asigurați-vă că în caseta de validare **Just-in-Time debugging** este prezent un marcaj de validare.

Opțiunea de depanare **OLE RPC** prezentă în această pagină este, de asemenea, foarte utilă la dezvoltarea aplicațiilor **COM** și **DCOM**, deoarece permite depanatorului să urmărească un apel de funcție către un alt program sau .dll extra-proces și să determine un alt

depanator să continue operația pentru acest proces. La revenirea din apelul la distanță, depanatorul original primește înapoi controlul, astfel că depanarea funcționează în rețea și între calculatoare diferite.

### VEDEȚI...

► Pentru mai multe amănunte despre **COM** și **OLE**, vedeți pagina 581.

## Diagnosticarea și execuția controlată

Una dintre cele mai folositoare facilități oferite de mediul de depanare din Visual C++ este execuția interactivă pas cu pas. Această facilitare vă permite să executați codul linie cu linie, inspectând concomitent conținutul variabilelor. Puteți, de asemenea, să fixați *puncte de întrerupere*, astfel încât programul să ruleze până la întâlnirea unui punct de întrerupere, oprindu-se în acel punct și permițându-vă să rulați pas cu pas până când optați pentru continuarea rulării.

### Puncte de întrerupere condiționale

Este posibilă asocierea cu punctele de întrerupere a unor condiții, ceea ce le face instrumente foarte puternice și sofisticate de depanare. Aceste condiții pot să testeze dacă anumite variabile ale programului se încadrează în diferite intervale sau pot determina ignorarea de un număr de ori a unui punct de întrerupere înainte de oprirea programului în linia respectivă.

Instrucțiunile de diagnosticare și aserțiunile sunt, la rândul lor, instrumente foarte utile pentru identificarea erorilor din programe. Instrucțiunile de diagnosticare vă permit să afișați prin program mesaje și variabile în cadrul ferestrei de ieșire (Output). Aserțiunile se folosesc pentru a opri execuția programului în cazul în care o condiție nu este TRUE atunci când dumneavoastră faceți aserțiunea că ar trebui să fie.

## Utilizarea macrodefiniției TRACE

Macrodefinițiile TRACE pot fi inserate în diverse locuri din program pentru a arăta că diferite secvențe de cod au fost rulate sau pentru a afișa conținutul variabilelor în locurile respective. Macrodefinițiile TRACE sunt compilate în configurația pentru depanare, iar efectele lor sunt afișate în fereastra de ieșire din pagina **Debug** atunci când programul este rulat prin intermediul depanatorului.

Nu vă preocupați de existența macrodefinițiilor TRACE la asamblarea versiunii finale, deoarece acestea sunt excluse automat din obiectul destinație.

Afișarea unor mesaje simple sau a conținutului variabilelor se face transmițând un șir de format ca prim parametru al macrodefiniției TRACE. Acest șir de format este identic cu cel pe care l-ați transmite într-un apel printf() sau CString::Format(). Puteți să specificați diferite coduri de formatare, cum ar fi %d pentru afișarea unui număr întreg, %x pentru afișarea unui număr hexazecimal sau %s pentru afișarea unui șir de caractere. Următorii parametri specificați vor trebui să corespundă cu ordinea codurilor de formatare. Spre exemplul, codul de mai jos:

```
int nMyNum = 60;
char* szMyString = "This is my String";
TRACE("Number = %d, or %x in hex and my string is: %s\n",
 nMyNum, szMyString);
// va duce la afișarea următoarei linii de diagnosticare:
// Number = 60, or 3c in hex and my string is
// This is my String
```

Listingul 27.1 ilustrează utilizarea macrodefiniției TRACE pentru afișarea conținutului unui vector înainte și după sortarea sa printr-un algoritm foarte ineficient, dar simplu.

Dacă vreți să verificați codul din listingul 27.1, puteți să folosiți AppWizard pentru a genera o infrastructură SDI simplă. Apoi adăugați codul deasupra funcției membru OnNewDocument() a clasei document și asigurați rularea sa prin adăugarea unui apel DoSort() în cadrul funcției OnNewDocument().

Pentru a vedea informațiile de diagnosticare, rulați aplicația prin intermediul depanatorului (efecuați un clic pe **Build**, selectați **Start Debug** și alegeți **Go** din submeniul afișat).

#### Combinatii de taste pentru depanare

O modalitate mai rapidă de a lansa procesul de depanare este apăsarea tastei F5. Aceasta va determina rularea programului prin intermediul depanatorului în cazul în care executabilul este actualizat, iar altfel vi se va propune generarea fișierelor necesare înainte de începerea depanării.

Trebuie să vă asigurați că fereastra de ieșire este vizibilă (efecuați un clic pe meniul **View** și selectați **Output**) atunci când este afișată fereastra paginată (aceeași ca la compilare). Asigurați-vă că este selectată pagina **Debug**.

#### LISTINGUL 27.1 LST30\_1.CPP – Un algoritm de sortare simplu pentru a ilustra tehnicile de depanare

```
1 void Swap(CUIntArray* pdwNumbers, int i) — ①
2 {
3 UINT uVal = pdwNumbers->GetAt(i);
4 pdwNumbers->SetAt(i, pdwNumbers->GetAt(i+1));
5 pdwNumbers->SetAt(i+1, uVal);
6 }
7
8 void DoSort()
9 {
10 CUIntArray arNumbers;
11 for(int i=0; i<10; i++) arNumbers.Add(1+rand()%100);
12
13 TRACE("Before Sort\n"); — ②
14 for(i=0; i<arNumbers.GetSize(); i++)
15 TRACE("[%d] = %d\n", i+1, arNumbers[i]);
16
```

```
17 BOOL bSorted;
18 do
19 {
20 bSorted = TRUE;
21 for(i=0; i<arNumbers.GetSize()-1; i++)
22 {
23 if (arNumbers[i] > arNumbers[i+1]) — ③
24 {
25 Swap(&arNumbers, i);
26 bSorted = FALSE;
27 }
28 }
29 } while(!bSorted);
30
31 TRACE("After Sort\n"); — ④
32 for(i=0; i<arNumbers.GetSize(); i++)
33 TRACE("[%d] = %d\n", i+1, arNumbers[i]);
34 }
```

① Funcția Swap() schimbă între ele două elemente consecutive de la un anumit indice din vector.

② Macrodefiniția TRACE este folosită pentru afișarea numerelor înainte de sortare.

③ Se testează fiecare pereche de numere și, dacă nu sunt ordonate, se schimbă între ele.

④ Macrodefiniția TRACE este folosită din nou pentru a afișa numerele după sortare.

Listingul 27.1 ordonează un vector de numere aleatoare (între 1 și 100), generat în linia 11. Liniile de la 13 la 15 afișează apoi conținutul vectorului înainte de sortarea sa, prin intermediul instrucțiunilor TRACE. Liniile de la 17 la 29 sortează vectorul schimbând perechi de elemente consecutive care nu sunt ordonate (prin apelul funcției Swap() din linia 25). Funcția Swap() (liniile de la 1 la 6) primește un pointer la vector și un indice, după care schimbă între ele cele două elemente începând de la acest indice.

După sortare, conținutul vectorului este afișat din nou în fereastra de ieșire, prin intermediul instrucțiunilor TRACE din liniile de la 31 la 33.

Informațiile de diagnosticare ale acestui program apar în fereastra de ieșire a mediului Developer Studio, ca în figura 27.3.

TABELUL 27.3 Ieșirea programului de sortare

| Înainte de sortare | După sortare |
|--------------------|--------------|
| [1] = 42           | [1] = 1      |
| [2] = 68           | [2] = 25     |
| [3] = 35           | [3] = 35     |
| [4] = 1            | [4] = 42     |

(continuare)

TABELUL 27.3 Continuare

| Înainte de sortare | După sortare |
|--------------------|--------------|
| [5] = 70           | [5] = 59     |
| [6] = 25           | [6] = 63     |
| [7] = 79           | [7] = 65     |
| [8] = 59           | [8] = 68     |
| [9] = 63           | [9] = 70     |
| [10] = 65          | [10] = 79    |

### Algoritmi de sortare

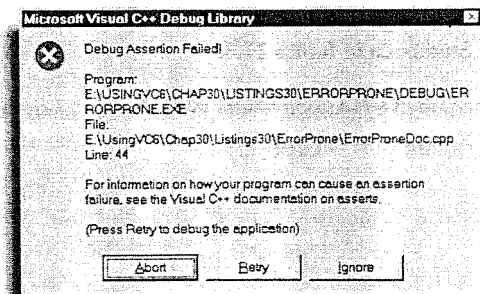
Deși această modalitate de sortare constituie un exemplu de depanare simplu și elocvent, este probabil unul dintre cel mai puțin eficienți algoritmi de sortare posibili. Dacă trebuie să implementați o sortare rapidă și eficientă, o posibilitate este algoritmul quick-sort (de sortare rapidă), disponibil prin intermediul funcției `qsort()` din biblioteca de execuție C++.

## Utilizarea macrodefinițiilor **ASSERT** și **VERIFY**

Macrodefiniția **ASSERT** se folosește pentru a asigura adevărul unor condiții. **ASSERT** primește un singur parametru, care este o expresie ce poate avea valoarea **TRUE** sau **FALSE**. Dacă expresia este **TRUE**, totul este în regulă. Dacă expresia este **FALSE**, programul se va opri și va afișa caseta de dialog Debug Assertion Failed (vedeți figura 27.7), care vă permite să suspendați execuția (**Abort**), să reluați evaluarea expresiei (**Retry**) sau să ignorați aserțiunea (**Ignore**). Sunt afișate, de asemenea, **Abort**, sesiunea de depanare ia sfârșit. **Retry** este, probabil, opțiunea cea mai utilă, deoarece va determina compilatorul să afișeze secvența de cod conținând macrodefiniția **ASSERT** care a eșuat, permițându-vă să determinați ce nu a mers. Dacă știți despre ce este vorba sau dacă nu vă pasă de eșecul aserțiunii, puteți să selectați **Ignore** și să continuați rularea programului, ceea ce ar putea să ducă ulterior la apariția unei erori și mai grave.

FIGURA 27.7

Caseta de dialog Debug Assertion Failed vă ajută să identificați erorile.



O utilizare tipică pentru **ASSERT** este asigurarea corectitudinii parametrilor primiți de o funcție. Spre exemplu, ați putea face mai sigură funcția `Sort()` (prezentată în listingul 27.1) prin verificarea parametrilor pe care-i primește. Pentru verificarea parametrilor, adăugați la începutul funcției `Sort()` următoarele macrodefiniții **ASSERT**:

```
ASSERT(pdwNumbers);
ASSERT(i >= 0 && i < 10);
```

Acestea asigură că pointerul la vectorul de numere nu este nul și că indicele primului element din perechea de schimbat este între 0 și 9. Dacă vreuna dintre aceste condiții nu este îndeplinită, va fi afișată caseta de dialog Debug Assertion Failed. Această tehnică vă ajută să identificați erorile produse de transmiterea unor parametri incorecți în apelurile de funcții. Este bine să verificați dacă valorile primite de fiecare funcție sunt adecvate, folosind în acest sens macrodefiniția **ASSERT**.

O altă macrodefiniție, **ASSERT\_VALID**, poate fi folosită împreună cu clasele derivate din **CObject**, așa cum sunt majoritatea claselor MFC. Această macrodefiniție efectuează o verificare mai riguroasă asupra obiectului și a conținutului său pentru a se asigura că starea sa este adecvată și validă. Verificarea unui obiect se face prin transmiterea unui pointer, ca aici:

```
ASSERT_VALID(pdwNumbers);
```

Tot o macrodefiniție de tip **ASSERT** este și **ASSERT\_KINDOF**, care este aplicată asupra claselor derivate din **CObject** pentru a confirma tipul acestora. Spre exemplu, puteți să verificați dacă obiectul indicat de un pointer aparține clasei reprezentare adecvate, astfel:

```
ASSERT_KINDOF(CYourSpecialView, pYView);
```

### Alte macrodefiniții de tip aserțiune

O macrodefiniție de tip aserțiune care se folosește mai puțin este **ASSERT\_POINTER**, care primește un pointer și tipul de dată/obiect indicat de acesta. **ASSERT\_POINTER** va verifica dacă pointerul este diferit de **NULL**, dacă indică o adresă validă de memorie și dacă memoria ocupată de întregul obiect este validă. O aserțiune înrudită este **ASSERT\_NULL\_OR\_POINTER**, care acceptă pointeri **NULL**, dar nu și pointeri la obiecte care ar ocupa o zonă de memorie invalidă pentru procesul curent.

În cazul în care obiectul indicat de pointer nu aparține clasei specificate sau unei clase derivate din aceasta, va fi afișată caseta de dialog Assertion Failed.

Trebuie să aveți grijă să nu plasați în cadrul macrodefinițiilor **ASSERT** cod necesar pentru funcționarea normală a programului, deoarece aceste macrodefiniții sunt ignorate în configurația pentru versiune finală. O sursă tipică de erori în cadrul versiunilor finale, erori greu de identificat, o reprezintă codul de genul următor:

```
int a = 0;
ASSERT(++a > 0);
if (a > 0) MyFunc();
```

În configurația pentru depanare, această secvență de cod va incrementa valoarea întreagă a în cadrul macrodefiniției **ASSERT**, după care va apela funcția `MyFunc()` din linia următoare,



pentru că a este mai mare decât zero. În momentul în care echipa dumneavoastră de vânzări este nerăbdătoare să prezinte noua aplicație, vă gândiți că totul merge bine pentru că în configurația pentru depanare n-a apărut nici o problemă. Așadar, asamblați versiunea finală și o transmiteți departamentului de vânzări, care o prezintă clientului și descoperă cum se blochează. S-a blocat pentru că operația ++a nu s-a mai efectuat – configurația pentru versiunea finală a eliminat liniile ASSERT.

Macrodefiniția VERIFY ajută la rezolvarea problemei de mai sus. VERIFY funcționează similar cu ASSERT, afișând în versiunea pentru depanare aceeași casetă de dialog Assertion Failed în cazul în care expresia transmisă este zero. Dar expresia este evaluată și în versiunea finală, numai că o valoare nulă nu mai afișează caseta de dialog. Prin urmare, veți folosi VERIFY atunci când vreți să evaluați o expresie întotdeauna și ASSERT atunci când vreți ca evaluarea să se efectueze doar în configurația pentru depanare. Consecința este că înlocuirea macrodefiniției ASSERT din exemplul precedent cu VERIFY, așa cum se vede mai jos, va permite funcționarea corespunzătoare a versiunii finale:

```
VERIFY(++a > 0);
```

De regulă, veți apela VERIFY pentru a testa valorile întoarse de funcții:

```
VERIFY(MyFunc() != FALSE);
```

## Utilizarea punctelor de întrerupere și a execuției controlate a programului

Utilizarea execuției controlate și a punctelor de întrerupere reprezintă probabil instrumentul cel mai eficient pentru identificarea majorității problemelor. Visual C++ pune la dispoziție mai multe tipuri de puncte de întrerupere, iar informațiile oferite la execuția controlată pot fi foarte sofisticate; sper, doar, că voi putea să vă ofer o imagine asupra puterii acestui instrument de depanare.

Elementul cheie al execuției controlate îl reprezintă punctele de întrerupere. Puteți să stabiliți un punct de întrerupere oriunde în cod, după care veți rula programul prin intermediul depanatorului. În momentul în care este atins punctul de întrerupere, în fereastra de editare este afișat codul conținând acest punct, iar dumneavoastră puteți să executați programul pas cu pas sau să continuați rularea normală.

Fixarea unui punct de întrerupere se face selectând linia de cod dorită (prin efectuarea unui clic în interiorul liniei din fereastra de editare) și apoi fie efectuând un clic pe butonul Breakpoint de pe mini-bara Build (vedeți figura 27.8), fie apăsând F9. Alternativ, se pot adăuga sau înlătura puncte de întrerupere mult mai sofisticate, efectuând un clic pe meniul Edit și selectând opțiunea Breakpoints pentru a afișa caseta de dialog Breakpoints (vedeți figura 27.9). La adăugarea unui punct de întrerupere, acesta este afișat sub forma unui mic cerc roșu în dreptul liniei specificate. Punctele de întrerupere nu pot fi stabilite decât pentru liniile valide de cod, așa că Developer Studio va deplasa uneori câte un punct de întrerupere în dreptul celei mai apropiate linii valide.

FIGURA 27.8

Adăugarea unui punct de întrerupere în cadrul codului, prin intermediul mini-barei Build sau apăsând tasta F9.

- 1 Compilare (Ctrl+F7)
- 2 Asamblare (F7)
- 3 Întreruperea asamblării (Ctrl+Break)
- 4 Rulare prin intermediul depanatorului (F5)
- 5 Adăugarea/înlăturarea unui punct de întrerupere

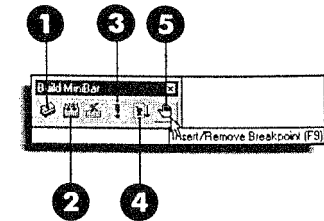
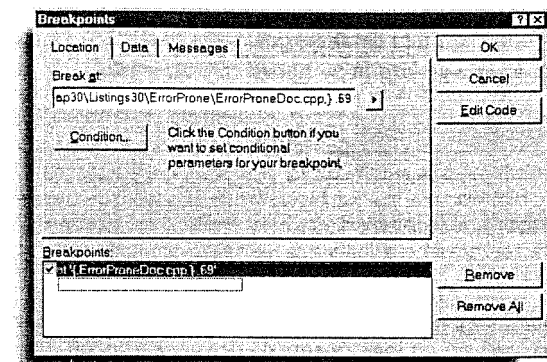


FIGURA 27.9

Adăugarea unui punct de întrerupere prin intermediul casei de dialog Breakpoints.



Adăugarea și înlăturarea unui punct de întrerupere se face efectuând clic pe butonul corespunzător de pe bara cu instrumente (cel care afișează o mână), eliminarea punctelor de întrerupere fiind posibilă și prin efectuarea unui clic pe unul din butoanele Remove sau Remove All din caseta de dialog Breakpoints. Puteți să dezactivați un punct de întrerupere fără a-l înlătura, efectuând clic pe marcajul de validare alăturat în cadrul listei afișate în caseta de dialog Breakpoints. Încă un clic aici va afișa din nou marcajul de validare și va activa la loc punctul de întrerupere.

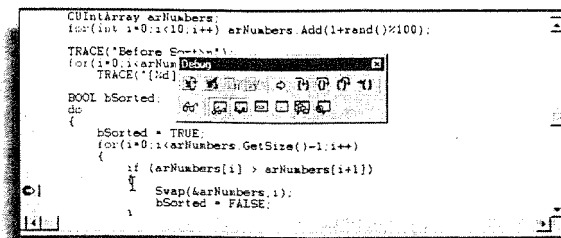
Odată stabilite punctele de întrerupere, rularea programului prin intermediul depanatorului se face selectând Build, Start Debug și Go. O altă posibilitate este folosirea unei scurtături, efectuând un clic pe butonul de pe bara cu instrumente (aflat pe bara cu instrumente Build Minibar, în stânga butonului pentru punctele de întrerupere – vedeți figura 27.8) sau apăsând tasta F5.

Programul va rula în mod obișnuit până când atinge punctul de întrerupere, moment în care se oprește și afișează o săgeată în dreptul liniei cu punctul de întrerupere. Acum puteți să folosiți bara cu instrumente de depanare (Debug) pentru a controla pașii de execuție, așa cum o ilustrează figura 27.10.



FIGURA 27.10

Debanatorul a oprit execuția la un punct de întrerupere, pregătit pentru execuția pas cu pas prin intermediul barei cu instrumente Debug.



În momentul în care programul este întrerupt în cadrul debanatorului, puteți să aflați valorile variabilelor prin simpla deplasare a cursorului deasupra acestora în cadrul ferestrei de editare. Respectiv valorile sunt afișate în dreptul cursorului, sub formă de etichetă flotantă. Informații mai detaliate pot fi afișate trăgând variabilele peste fereastra Watch, așa cum vom discuta pe larg în secțiunea care urmează.

#### Viteza la execuția pas cu pas și până la cursor

Dacă efectuați operații de execuție controlată, așa cum sunt execuția pas cu pas și până la cursor, s-ar putea să observați o mică întârziere, mai ales în cazul în care debanatorul trebuie să execute o buclă mai mare. Acest lucru demonstrează natura semi-interpretată a debanatorului, care trebuie să se oprească după fiecare instrucțiune pentru a verifica dacă a fost atinsă linia de cod pe care ați specificat-o.

Execuția pas cu pas a codului poate fi efectuată prin intermediul celor patru butoane cu acolade de pe bara cu instrumente de depanare (Step Into, Step Over, Step Out, Run to Cursor) sau efectuând un clic pe meniul **Debug** și selectând opțiunile de execuție dorite. Aceste opțiuni sunt prezentate în tabelul 27.4. Ele sunt disponibile în meniul **Debug** și pe bara cu instrumente de depanare.

TABELUL 27.4 Opțiunile disponibile pentru execuția controlată

| Buton/opțiune de execuție | Combinăție de taste | Efect                                                                                                                                                                                                                 |
|---------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Step Into                 | F11                 | Debanatorul execută linia curentă, iar dacă aceasta conține un apel de funcție, va pătrunde în interiorul ei.                                                                                                         |
| Step Over                 | F10                 | La fel ca Step Into, cu deosebirea că dacă linia conține un apel de funcție, debanatorul execută această funcție la viteza normală și apoi se oprește la revenire, dând impresia de execuție dintr-o dată a funcției. |
| Step Out                  | Shift+F11           | Debanatorul execută restul funcției la viteza normală și se oprește la revenire, în cadrul funcției apelante.                                                                                                         |
| Run to Cursor             | Ctrl+F10            | Debanatorul execută programul până la poziția curentă a cursorului. Această poziție se stabilește efectuând un clic pe linia la care vreți să se oprească execuția.                                                   |

#### Buton/opțiune de execuție

#### Combinăție de taste

#### Efect

|                    |               |                                                                                                                                                                         |
|--------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Go                 | F5            | Continuă rularea la viteza normală a programului până la întâlnirea următorului punct de întrerupere.                                                                   |
| Stop Debugging     | Shift+F5      | Această opțiune oprește debanatorul și revine în modul de editare.                                                                                                      |
| Restart            | Ctrl+Shift+F5 | Această opțiune reia de la început execuția programului, oprindu-se chiar la prima linie de cod.                                                                        |
| Break Execution    |               | Această opțiune întrerupe execuția la viteza normală a unui program.                                                                                                    |
| Apply Code Changes | Alt+F10       | Această opțiune vă permite să compilați codul după efectuarea unor modificări în cadrul unei sesiuni de depanare, continuând apoi depanarea de acolo de unde ați rămas. |

Prin intermediul opțiunilor de mai sus puteți să urmăriți execuția programului și să vizualizați valorile variabilelor pe măsură ce acestea sunt manipulate prin cod. Săgeata galbenă din fereastra de editare indică întotdeauna următoarea instrucțiune de executat.

Secțiunile care urmează prezintă câteva dintre ferestrele pe care le puteți folosi atunci când execuția programului este întreruptă în cadrul debanatorului.

### Utilizarea facilității de editare și continuare

O excelentă facilităate care apare nouă în Visual C++ 6.0 este facilităatea de editare și continuare. Aceasta vă oferă posibilitatea să modificați sau să editați codul în timp ce execuția este întreruptă în cadrul debanatorului. După începerea editării, veți putea observa că opțiunea **Apply Code Changes** din meniul **Debug** devine activată (ca și butonul corespunzător – – de pe bara cu instrumente de depanare). În consecință, puteți să selectați această opțiune (sau butonul de pe bara cu instrumente ) pentru a compila codul modificat și a continua depanarea codului acum actualizat. Prin intermediul acestei facilități, puteți să corectați erorile în timpul depanării și să continuați rularea de acolo de unde ați rămas, cu aceleași valori ale variabilelor, ceea ce poate fi foarte util în cazul depanării unor programe mari și complexe.

### Urmărirea variabilelor din program

Figura 27.11 înfățișează ferestrele de urmărire (Watch) și de variabile (Variables). Aceste ferestre afișează valori de variabile atunci când execuția este întreruptă în cadrul debanatorului. Afișarea ferestrelor se face efectuând un clic pe meniul **View** și selectându-le din submeniul **Debug Windows** sau efectuând clic pe butoanele și , aflate pe bara cu instrumente.

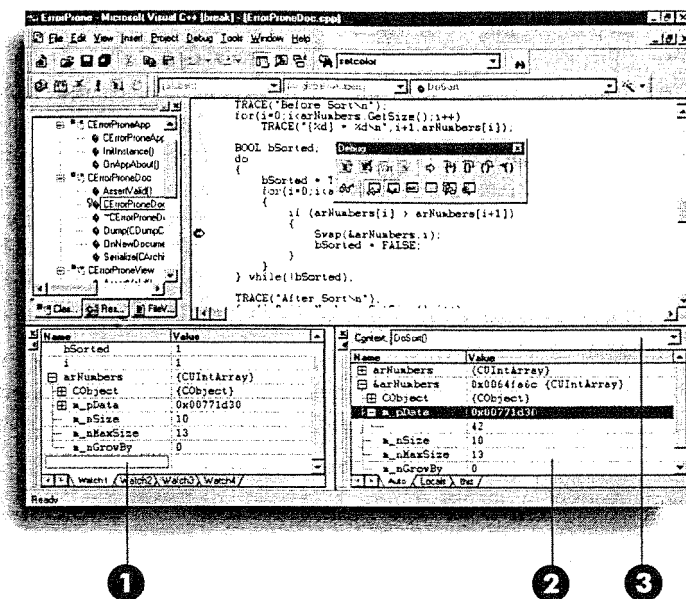
Fereastra de variabile afișează întotdeauna variabilele locale ale funcției selectate în caseta combinată **Context** din partea superioară a ferestrei. Pentru a afla cum s-a ajuns în funcția curentă, derulați lista casetei combinate și veți vedea toate funcțiile care au fost apelate.

Aceasta este *stiva de apeluri* și reprezintă contextul curent al programului, afișând lista funcțiilor care s-au apelat pentru a ajunge la funcția care se execută atunci când s-a întrerupt execuția. Dacă selectați o altă funcție, vor fi afișate variabilele locale din funcția respectivă.

FIGURA 27.11

Fereastra de urmărire afișează valorile variabilelor în timpul depanării.

- 1 Fereastra de urmărire.
- 2 Fereastra de variabile.
- 3 Caseta combinată Context.



Puteți să expandați pointerii la obiecte care sunt afișați, efectuând un clic pe semnul plus alăturat numelui de pointer. Pointerul special `this` din C++ este afișat întotdeauna pentru funcțiile membru ale claselor, putând fi expandat pentru a vizualiza toate variabilele membru ale obiectului curent.

Fereastra de urmărire vă permite să specificați variabile introducându-le numele de la tastatură sau trăgându-le din fereastra de editare (după selectarea și evidențierea lor cu ajutorul cursorului). Valorile înscrise în variabilele afișate sunt prezente atât timp cât respectivele variabile se află în domeniile lor de existență (adică până când devin irelevante pentru funcția depanată curent).

În fereastra de urmărire pot fi specificate și conversii simple sau indici de vector, rafinând astfel valorile afișate. Efectuarea unui clic cu butonul drept comută între afișarea valorilor sub formă zecimală și sub formă hexazecimală. Pe măsură ce executați liniile de cod, valorile afișate în ferestrele de urmărire și de variabile sunt actualizate corespunzător, permițându-vă să urmăriți modul în care programul manipulează variabilele.

## Alte ferestre pentru depanare

Există și alte ferestre pentru depanare, disponibile prin efectuarea unui clic pe meniul **View** și selectarea lor din submeniul **Debug Windows** sau, alternativ, prin efectuarea unui clic

pe butoanele corespunzătoare aflate în partea dreaptă a barei cu instrumente de depanare. Aceste ferestre sunt:

- **Quick Watch.** Dacă efectuați clic pe o variabilă din cod și selectați **Quick Watch** sau apăsați Shift+F9, va fi afișat conținutul variabilei respective. Există și posibilitatea de a introduce variabile direct, efectuând apoi clic pe butonul **Add Watch** pentru a le trimite în fereastra de urmărire.

### Evaluarea expresiilor cu Quick Watch

Fereastra **Quick Watch** poate fi utilizată și pe post de calculator, permițându-vă să introduceți expresii matematice și logice, incluzând și variabilele din program evaluate la valorile curente. În momentul în care apăsați pe butonul **Recalculate**, rezultatul expresiei este afișat în caseta **Current Value**.

- **Registers.** Fereastra **Registers** afișează valorile curente din setul de registre ale procesorului. Nu este o facilităate atât de utilă, mai puțin în cazul în care operați la nivelul codului mașină sau de asamblare.
- **Memory.** Fereastra **Memory** afișează conținutul memoriei din spațiul de adrese al aplicației, pe coloane fiind afișate adresele, valorile sub formă hexazecimală și caracterele corespunzătoare pentru fiecare octet. Puteți să optați pentru afișarea valorilor sub formă de **Byte**, **Short** sau **Long**, efectuând clic cu butonul drept și selectând opțiunile corespunzătoare din meniul de context.
- **Call Stack.** Fereastra **Call Stack** afișează lista funcțiilor care au fost apelate pentru a ajunge la funcția curentă, împreună cu valorile parametrilor transmiși fiecărei funcții. Această facilităate poate fi foarte utilă pentru a urmări fluxul de execuție a programului care a dus la apelul unei anumite funcții. Prin efectuarea unui dublu clic pe o funcție din listă, într-o fereastră de editare va fi afișată secvența de cod care conține apelul funcției respective.

Acolo unde nu este disponibil codul sursă, funcțiile sunt afișate ca aici:

```
KERNEL32! Bff88f75()
```

Dacă efectuați un clic pe o astfel de intrare, codul afișat va fi în limbaj de asamblare, nu în C++.

- **Dissassembly.** Prin selectarea opțiunii de meniu **Dissassembly** sau a butonului corespunzător de pe bara cu instrumente, puteți să comutați între afișarea mixtă de cod C++ și în limbaj de asamblare și afișarea exclusiv de cod C++. Acolo unde nu există cod sursă disponibil, va fi afișat doar codul în limbaj de asamblare.

### Execuția pas cu pas a codului în limbaj de asamblare

Atunci când este afișată fereastra **Dissassembly**, depanatorul execută instrucțiunile cu instrucțiunile codul în limbaj de asamblare. Codul în limbaj de asamblare și codurile de operație corespunzătoare reprezintă octeții brute ai programului, formând instrucțiunile pentru procesor. Execuția pas cu pas a codului în limbaj de asamblare este înecată și laborioasă, dar oferă răspunsuri exacte care explică de ce programul nu se comportă așa cum vă așteptați. De multe ori, aceasta este singura soluție pentru identificarea unui cod eronat având drept cauză erori ale însuși compilatorului, situație care poate apărea uneori la compilarea cu optimizări a unui cod sursă complex.

## Alte instrumente pentru depanare

Pe lângă instrumentele pentru depanare integrate, există alte câteva instrumente care nu sunt integrate, dar sunt foarte utile. Acestea pot fi apelate efectuând un clic pe meniul **Tools** și selectând opțiunea corespunzătoare.

În general, aceste instrumente vă permit să urmăriți aspecte legate de operarea în sistem, așa cum sunt mesajele Windows, procesele în execuție sau obiectele OLE înregistrate, îmbogățind astfel informațiile disponibile la depanarea aplicației.

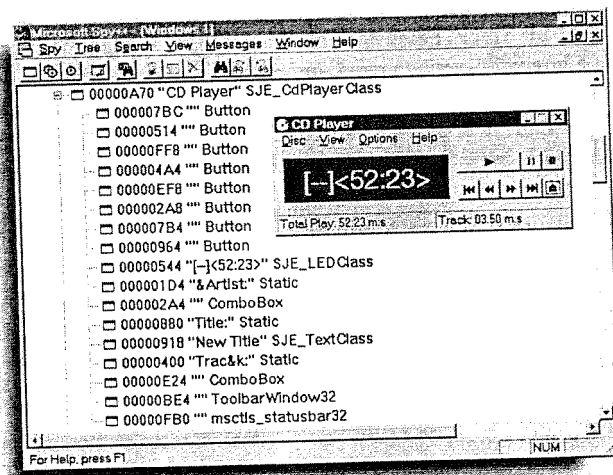
## Utilizarea programului Spy++

Spy++ este, fără îndoială, unul dintre cele mai utile instrumente. Prin intermediul Spy++ puteți să vizualizați relațiile ierarhice dintre ferestrele părinte și copil, pozițiile și opțiunile de stil ale ferestrelor și clasele de bază ale acestora. Puteți, de asemenea, să urmăriți mesajele trimise către ferestre.

După ce lansați Spy++, acesta afișează toate ferestrele de pe suprafața de lucru, ferestrele înrudite și clasa de bază pentru fiecare obiect fereastră (vedeți figura 27.12). Reprezentarea înfățișată în figura 27.12 a fost derulată pentru a afișa informații despre programul standard de redare a CD-urilor din Windows. Spy++ vă arată toate butoanele și casetele combinate, ele însele ferestre subordonate ferestrei principale a programului.

FIGURA 27.12

Reprezentarea inițială din Spy++, conținând ferestrele de pe suprafața de lucru și înfățișând aici datele referitoare la CD Player.



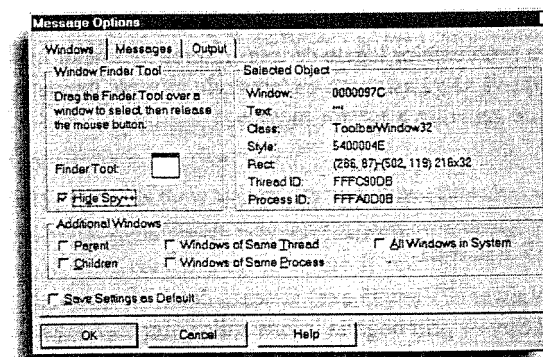
Dacă efectuați un clic pe meniul **Spy**, veți întâlni următoarele opțiuni:

- **Messages.** S-ar putea să descoperiți că reprezentarea Messages este probabil una dintre cele mai utile facilități, permițându-vă să urmăriți mesajele trimise oricăror ferestre (inclusiv celor din propria aplicație). Este posibilă, totodată, filtrarea mesajelor, astfel încât să nu vă pomeniți cu o avalanșă de mesaje generate de simpla deplasare a mouse-ului.

Pentru a urmări mesajele, selectați această opțiune și va fi afișată caseta de dialog Message Options, prezentată în figura 27.13. În continuare, puteți să trageți instrumentul de localizare peste orice fereastră din sistem, detaliile corespunzătoare fiind afișate odată cu deplasarea instrumentului. În plus, Spy++ evidențiază fereastra selectată, astfel că puteți să identificați ferestrele cadru și client. Odată găsită fereastra care vă interesează, dați drumul instrumentului. Acum puteți să folosiți celelalte pagini ale casetei de dialog pentru a stabili opțiunile de filtrare și de formatare a rezultatelor afișate. După ce ați terminat, efectuați un clic pe **OK** pentru a închide caseta Message Options.

FIGURA 27.13

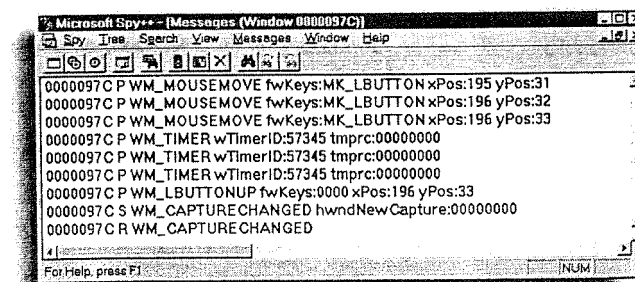
Identificarea ferestrelor cu ajutorul instrumentului de localizare din caseta Message Options a programului Spy++.



Cele afișate în figura 27.14 sunt mesajele generate la utilizarea barei cu instrumente a unei aplicații SDI obișnuite. Precum vedeți, în lipsa unei filtrări sunt afișate o mulțime de mesaje privind deplasarea mouse-ului și urmărirea cursorului, dar puteți remarca și familiarul mesaj **WM\_LBUTTONDOWN**, împreună cu parametri de poziție asociați.

FIGURA 27.14

Mesajele generate la utilizarea unei bare cu instrumente, jurnalizate de Spy++.



- **Windows.** Reprezentarea Windows este cea prezentată în figura 27.12, având organizarea și structura suprafeței de lucru din Windows. Dacă efectuați un dublu clic pe una dintre ferestre, va fi afișat un catalog de proprietăți care conține informațiile de poziție și opțiunile de stil ale ferestrei selectate. Pentru actualizarea acestor informații, efectuați un clic pe meniul **Windows** și selectați **Refresh**.

- **Processes.** Reprezentarea Processes vă prezintă toate programele aflate în execuție. Intrările corespunzătoare pot fi expandate pentru a afișa firele de execuție și ferestrele asociate acestora.
- **Thread.** Opțiunea Threads afișează aceleași informații, dar fără subordonarea față de procese, astfel că puteți să vedeți fiecare fir de execuție aflat în rulare și ferestrele pe care acesta le deține.

Spy++ este prea complex pentru a putea fi discutat aici sub toate aspectele, dar rămâne un instrument de neînlocuit care vă ajută să înțelegeți organizarea ierarhiei de ferestre și a sistemului de mesaje. V-ați putea îmbogăți cunoștințele prin simpla examinare a unor aplicații comerciale cu ajutorul Spy++. Este, totodată, un instrument excelent pentru depanarea problemelor legate de transmiterea mesajelor în cadrul unei aplicații, permițându-vă să vă asigurați că ferestrele primesc mesajele corespunzătoare și ilustrând secvența de generare a acestor mesaje.

#### VEDEȚI...

- Pentru mai multe detalii privind mesajele din Windows, vedeți pagina 75.

## Process Viewer

Process Viewer (PView95.exe) afișează detalii despre toate procesele în execuție, întrecând Spy++ prin bogăția informațiilor oferite. Această aplicație poate fi lansată din cadrul meniului **Start** al sistemului Windows, aflându-se în submeniul **Microsoft Visual Studio 6.0 Tools** (sau similar) accesibil din meniul **Programs**. Programul afișează toate procesele care rulează în cadrul sistemului, permițându-vă să le ordonați prin efectuarea unui clic pe oricare dintre antetele de coloană. Prin efectuarea unui clic pe un proces vor fi afișate toate firele de execuție ale acestuia. Figura 27.15 înfățișează programul Process Viewer, în cadrul său fiind selectată aplicația Developer Studio (MSDEV.EXE) și fiind afișate toate firele de execuție ale acesteia.

FIGURA 27.15

Process Viewer înfățișează MSDEV.EXE și firele sale de execuție.

| Process           | PID       | Base Priority | Num. Threads | Type   | Full Path                        |
|-------------------|-----------|---------------|--------------|--------|----------------------------------|
| PVIEW95.EXE       | FFFB553   | 8 (Normal)    | 1            | 32-Bit | C:\PROGRAM FILES\DEVSTUDIO\V...  |
| WINWORD.E...      | FFFD2517  | 8 (Normal)    | 3            | 32-Bit | D:\MICROSOFT\OFFICE\WINWORD.E... |
| ESPORPRO...       | FFFA0D... | 8 (Normal)    | 1            | 32-Bit | E:\USINGVC\CHAP30\LISTING530...  |
| <b>MSDEV95.XE</b> | FFFA4747  | 8 (Normal)    | 7            | 32-Bit | C:\PROGRAM FILES\DEVSTUDIO\S...  |
| CDPLAYER.E...     | FFFB194B  | 8 (Normal)    | 1            | 32-Bit | C:\WINDOWS\CDPLAYER.EXE          |

| TID      | Owning PID | Thread Priority  |
|----------|------------|------------------|
| FFFC3743 | FFFA4747   | 8 (Normal)       |
| FFFA474F | FFFA4747   | 7 (Below Normal) |
| FFFBF95B | FFFA4747   | 9 (Above Normal) |
| FFFB10D3 | FFFA4747   | 8 (Normal)       |
| FFFAA51F | FFFA4747   | 7 (Below Normal) |
| FFFC3367 | FFFA4747   | 8 (Normal)       |
| FFFD7AE3 | FFFA4747   | 8 (Normal)       |

## OLE/COM Object Viewer

Instrumentul OLE/COM Object Viewer vă prezintă toate obiectele OLE/COM înregistrate în sistem, printre care controalele ActiveX, bibliotecile de tipuri, obiectele încapsulabile, obiectele de automatizare și multe alte categorii.

Puteți chiar să creați instanțe ale diverselor obiecte și să examinați în detaliu interfețele acestora. OLE/COM Object Viewer este foarte util în cazul în care dezvoltați o aplicație OLE/COM sau atunci când încercați să localizați un control ActiveX mai obscur.

#### VEDEȚI...

- Pentru a vedea cum se utilizează controalele ActiveX în cadrul aplicațiilor, vedeți pagina 186.
- În legătură cu crearea controalelor ActiveX, vedeți pagina 605.
- Pentru mai multe detalii despre COM și OLE, vedeți pagina 581.

## MFC Tracer

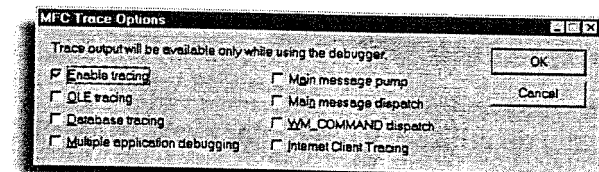
Instrumentul MFC Tracer, înfățișat în figura 27.16, vă permite să opriți procesul normal de diagnosticare sau să adăugați elemente de diagnosticare specifice sistemului Windows pe lângă cele generate în mod obișnuit de programul dumneavoastră. La apelarea acestui instrument sunt afișate o serie de casete de validare pe care le puteți selecta sau deselecta pentru a stabili opțiunile de diagnosticare respective.

Puteți să urmăriți mesajele Windows, mesajele bazelor de date, mesajele OLE și multe alte informații de diagnosticare, toate acestea ajutându-vă la identificarea unor probleme obscure. Mesajele vor fi generate de codul MFC, în concordanță cu diversele opțiuni selectate.

Puteți chiar să dezactivați informațiile de diagnosticare obișnuite pe care le generează aplicația, deselectând opțiunea **Enable Tracing**.

FIGURA 27.16

Opțiunile oferite de instrumentul MFC Tracer.



# Utilizarea pachetelor API și SDK

**Modul de utilizare a diferitelor pachete API și SDK aflate la dispoziția programatorilor în Visual C++**

**Crearea unor aplicații de sunet și grafică foarte rapide cu ajutorul DirectX**

**Trimiterea și recepționarea mesajelor e-mail prin intermediul arhitecturii MAPI**

**Utilizarea interfeței pentru controlul multimedia în scopul manipulării clipurilor audio și video**

## Introducere în API și SDK

Pachetele API (Application Programming Interface – interfață pentru programarea aplicațiilor) reprezintă colecții de funcții stocate de regulă în biblioteci cu legare dinamică (.dll) sau statică (.lib). Aceste pachete vă ajută la implementarea unor tipuri specifice de funcționalități, cum ar fi cele legate de grafică, sunet, acces la baze de date sau comunicații, punând la dispoziție un nivel intermediar de cod între aplicație și componentele hardware.

Există multe pachete API diferite care simplifică o gamă largă de probleme de programare, unificându-le sub forma unor mulțimi standard de funcții ce pot fi apelate. Pot fi găsite cărți întregi dedicate folosirii diferitelor pachete API individuale, așa că în capitolul de față voi încerca doar să vă prezint câteva exemple de utilizare a unor pachete mai cunoscute.

În dezvoltarea modernă a aplicațiilor, cu ajutorul tehnologiilor COM și OLE, multe dintre pachetele API sunt implementate sub forma unor obiecte COM, ceea ce ajută foarte mult la rezolvarea problemelor de versiune care se datorează existenței mai multor versiuni ale aceluiași .dll.

Pachetele SDK (Software Development Kits – kit pentru dezvoltarea aplicațiilor) constă din pachete API însoțite de documentație și exemple de programe. Aceste pachete vă ajută la dezvoltarea unor tipuri specifice de aplicații, exemple fiind Game SDK, pentru dezvoltarea jocurilor, sau OLE DB SDK, pentru dezvoltarea aplicațiilor OLE de baze de date.

În general, funcțiile API nu sunt incluse printre bibliotecile legate în mod implicit la proiect, fiind necesară specificarea bibliotecilor corespunzătoare. De obicei, toate prototipurile de funcții și macrodefinițiile API necesare se pot adăuga prin includerea unui singur fișier de antet .h, specific pachetului API în cauză.

### Obținerea pachetelor API și SDK

Visual C++ nu conține toate pachetele API și SDK, dar majoritatea acestora pot fi descărcate de pe situl de Web al Microsoft, la adresa [www.microsoft.com](http://www.microsoft.com), de regulă gratis. Chiar dacă pachetul API sau SDK pe care doriți să-l folosiți este furnizat împreună cu Visual C++, de cele mai multe ori este bine ca, înainte de a începe, să verificați dacă nu există o versiune mai nouă.

### VEDEȚI...

➤ Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.

## Implementarea de sunet și grafică rapide cu DirectX

Pachetele API DirectX au fost create pentru a rezolva problema accesului rapid la componentele hardware video și audio. Deși Windows oferă funcții standard sofisticate pentru grafică și sunet, acestea nu s-au dovedit suficient de flexibile sau de rapide pentru programatorii care au încercat să dezvolte jocuri rapide de acțiune sau aplicații de prezentare grafică. O bună bucată de vreme, singura soluție era scrierea aplicației pentru DOS, pierzând astfel avantajele aduse de Windows, printre care independența de hardware, informațiile de configurare și alte servicii oferite.

**Dificultățile implicate de grafica de înaltă viteză**

Problema principală care apare la dezvoltarea aplicațiilor care folosesc grafică animată ține de puterea brută a procesorului. Pentru a păcăli ochiul uman și a-i da impresia de mișcare continuă, și nu de secvență de imagini, o aplicație trebuie să afișeze circa 24 de cadre pe secundă. La o rezoluție de 800x600 cu 256 de culori sunt afișați 480.000 de pixeli. Pentru a schimba culoarea fiecărui pixel de 24 de ori pe secundă înseamnă că trebuie să efectuați 11.520.000 de operații într-o secundă. Afișarea fiecărui pixel ar necesita în jur de 12 cicluri de ceas ale procesorului, ceea ce înseamnă 138.240.000 de cicluri de ceas doar pentru înprospătarea ecranului. În cazul unui procesor modern la 266Mhz, 50% din timpul acestuia ar fi ocupat doar de afișarea pe ecran. Apoi, în restul de 50% din timp, va trebui să efectuați toate calculele complexe și laborioase legate de scena 3D, maparea texturilor, scalarea bitmap-urilor și interacțiunea cu utilizatorul.

Microsoft a acoperit această necesitate prin intermediul unui set de pachete API care oferă în continuare independența de hardware, dar permit un acces mult mai rapid și mai direct la dispozitivele hardware video, audio și de intrare. Aceste pachete API sunt

DirectSound, DirectDraw, Direct3D, DirectPlay, DirectInput și DirectSetup; împreună, ele formează Game SDK, un pachet care s-a dovedit nemaipomenit de popular printre producătorii din solicitanta industrie a jocurilor și a graficii de prezentare.

Toate aceste pachete API sunt implementate prin intermediul modelului componentelor obiectuale (COM), ceea ce înseamnă că funcțiile oferite sunt grupate în interfețe COM ale obiectului care le implementează.

Bibliotecile DLL necesare acestor pachete API nu sunt instalate odată cu Windows, dar sunt disponibile gratuit și însoțesc majoritatea jocurilor actuale. Ele pot fi descărcate și de pe situl de Web al Microsoft, la adresa [www.microsoft.com/directx](http://www.microsoft.com/directx).

**VEDEȚI...**

➤ Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.

**Utilizarea interfeței DirectSound**

Interfața DirectSound vă permite să accesați direct placa de sunet pentru a genera sunet pe baza conținutului unui *buffer pentru semnal de undă*. Acest buffer nu este decât un segment de memorie care poate fi citit direct de placa de sunet, aceasta traducând valorile găsite în nivele de sunet. În tehnica cea mai simplă de codificare, fiecare octet din buffer poate reprezenta oricare dintre 256 de nivele de amplitudine a sunetului, corespunzând direct vibrației membranelor difuzoarelor care sunt conectate la placa de sunet.

Prin înscrierea în buffer a unor valori care reprezintă un semnal sinusoidal, este produs un sunet pur de o anumită tonalitate. Dacă modificați amplitudinea sinusoidale în diferite locuri din buffer, semnalul de undă este modulată și produce diferite efecte.

Acest buffer pentru semnal de undă se numește buffer primar DirectSound. În mod normal nu veți crea direct un buffer primar; de regulă veți crea buffere secundare, iar la redarea acestor buffere secundare, ele vor fi mixate automat pentru a genera bufferul primar.

Prin intermediul acestui mecanism de mixare a mai multor buffere secundare, este foarte ușor să produceți efecte sonore foarte complexe. Figura 28.1 înfățișează ieșirea unui

exemplu de program care mixează două buffere secundare foarte diferite. Bufferul din partea superioară este o undă sinusoidală atenuată (sunet pur), iar bufferul din mijloc conține valori aleatoare (*zgomot alb* sau fâșâit) de amplitudine crescătoare. Semnalul de undă din partea inferioară reprezintă conținutul bufferului primar care rezultă din redarea celor două buffere, astfel mixate. Efectul la redare va fi un sunet pur care se pierde repede în *zgomot alb*. Puteți să mixați sunete cu alte sunete, cu unde pătrate, cu semnale dinte de fierăstrău sau cu orice alt conținut de buffer, inclusiv eșantioane de sunet (înregistrate), producând astfel semnalul sonor dorit.

Primul lucru pe care trebuie să-l faceți pentru a utiliza DirectSound este să creați o interfață IDirectSound, fie printr-un apel direct al funcției CoCreateInstance() (vedeți capitolul 25, „Despre programarea OLE și COM”), fie prin intermediul funcției ajutoare DirectSoundCreate().

**Utilizarea funcției DirectSoundCreate()**

Dacă folosiți DirectSoundCreate(), trebuie să specificați biblioteca dsound.lib în caseta **Object/Library Modules** din pagina **Link** a casei de dialog **Project Settings**. Dacă apeleți la soluția cu CoCreateInstance(), nu veți mai avea nevoie de această bibliotecă, însă va trebui să declarați valorile CLSID (identificator de clasă) și IID (identificator de interfață) necesare.

DirectSoundCreate() necesită trei parametri; primul este un pointer la un GUID (identificator global unic) care identifică dispozitivul de sunet pe care doriți să-l folosiți. Acești identificatori pentru dispozitivele de sunet pot fi determinați cu ajutorul unei funcții cu apel invers și al funcției DirectSoundEnumerate(). Alternativ, puteți transmite NULL pentru a folosi dispozitivul de sunet implicit. Al doilea parametru este un pointer care indică pointerul la interfața obiectului DirectSound, fiind folosit pentru a transmite aplicației pointerul la interfață. Cel de-al treilea parametru este de regulă NULL, mai puțin în cazul în care trebuie să utilizați tehnica de agregare COM.

În cazul în care funcția reușește, pointerul furnizat indică o interfață IDirectSound, care poate fi eliberată prin apelul funcției Release(), asemeni oricărei interfețe COM.

Odată ce aveți la dispoziție o interfață IDirectSound, trebuie imediat să stabiliți un nivel de cooperare. Multe dintre pachetele API din DirectX sunt caracterizate de un nivel de cooperare care specifică modul de partajare între aplicații a dispozitivului hardware folosit. Acest nivel poate fi stabilit prin apelul funcției SetCooperativeLevel() a interfeței IDirectSound, transmitând indicatorul Windows corespunzător propriei aplicații. Trebuie, de asemenea, să transmiteți o valoare indicator pentru nivelul de cooperare dorit. În mod normal se folosește nivelul de cooperare DSSCL\_NORMAL, care permite o cooperare totală cu celelalte aplicații.

În continuare puteți să creați bufferele de sunet secundare, folosind funcția CreateSoundBuffer() a interfeței IDirectSound. CreateSoundBuffer() primește trei parametri. Primul este o structură DSBUFFERDESC care descrie tipul de buffer pe care doriți să-l creați. Al doilea este un pointer care indică un pointer la interfața IDirectSoundBuffer pentru noul buffer. Al treilea parametru este de regulă NULL, mai puțin în cazul în care este nevoie de agregare COM.



Structura `DSBUFFERDESC` este definită astfel:

```
typedef struct _DSBUFFERDESC{
 DWORD dwSize;
 DWORD dwFlags;
 DWORD dwBufferBytes;
 DWORD dwReserved;
 LPWAVEFORMATEX lpwfxFormat;
} DSBUFFERDESC, *LPDSBUFFERDESC;
```

Variabilele membru ale acestei structuri au următoarele semnificații:

- `dwSize` reprezintă dimensiunea structurii (care poate fi determinată cu ajutorul operatorului `sizeof()`).
- `dwFlags` vă permite să stabiliți mai multe opțiuni care controlează elemente precum stereofonia, balansul și volumul sunetului. Un set bun de valori implicite poate fi stabilit prin specificarea indicatorului `DSBCAPS_CTRLDEFAULT`.
- `dwBufferBytes` reprezintă dimensiunea în octeți a bufferului. Pentru eşantioane lungi sau pentru sunete complexe ați putea avea nevoie de un buffer mai mare; pentru sunete scurte sau care se repetă puteți folosi un buffer mai mic. Bufferele pot fi redade ciclic, conținutul lor fiind astfel refolosit pentru producerea unor efecte repetitive de durată.
- `lpwfxFormat` indică o structură `WAVEFORMATEX`, care conține informații ce specifică plăcii de sunet modul de redare a sunetului.

Structura `WAVEFORMATEX` reține informații precum viteza la care trebuie redat bufferul și câți octeți din buffer reprezintă o valoare de sunet. Tot aici este înscrisă și tehnica de redare a sunetului, care este dependentă de hardware, dar tehnica `WAVE_FORMAT_PCM`, prezentată mai devreme, este suportată de majoritatea plăcilor de sunet.

În momentul în care aveți un buffer secundar, puteți stabili conținutul acestuia fie încărcând un fișier `.wav`, fie prin cod. În ambele cazuri, accesul la buffer este controlat prin apelul unei funcții `Lock()` a interfeței `IDirectSoundBuffer`, care întoarce adresa și dimensiunea bufferului. În continuare veți putea să compuneți în buffer semnalul de undă dorit, apelând apoi funcția `Unlock()` pentru a debloca bufferul.

#### Blocarea și deblocarea bufferului

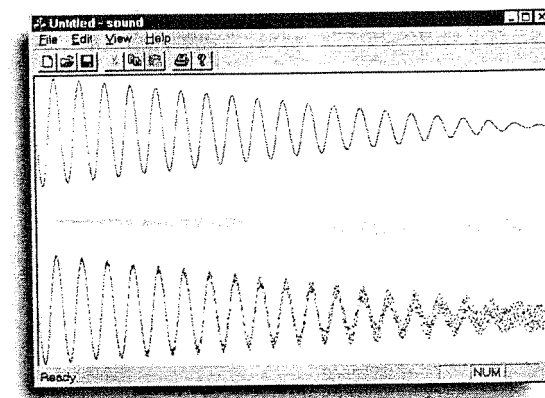
Bufferul de sunet nu trebuie blocat pentru perioade lungi de timp; în caz contrar, este posibil ca sunetul redat curent să necesite conținutul blocat al bufferului, caz în care este generat un zgomot aleator. Unele plăci de sunet dispun de buffere de memorie proprii, care nu sunt accesibile în spațiul de adrese principal al procesorului. În acest caz, funcțiile `Lock()` și `Unlock()` se ocupă și de transferul datelor de la și înapoi către bufferul de pe placa de sunet, trecând prin memoria sistemului pentru a permite actualizarea lor de către program.

După înscrierea conținutului bufferului, redarea sa se efectuează apelând funcția `Play()` a interfeței `IDirectSoundBuffer`, aceasta necesitând trei parametri. Primii doi sunt rezervați și trebuie fixați la 0. Al treilea poate fi `DSBPLAY_LOOPING`, pentru a determina redarea ciclică a bufferului, sau zero, pentru a reda bufferul o singură dată. Operația de redare poate fi întreruptă prin apelul funcției `Stop()`.

Listingul 28.1 ilustrează modul în care puteți folosi `DirectSound` pentru a crea un sintetizator de sunet simplu, dar elocvent, controlat de tastatură prin intermediul clasei reprezentare a unei aplicații SDI obișnuite. Programul creează și afișează - cu roșu și verde - conținutul a două buffere secundare reprezentând o sinusoidă atenuată și, respectiv, un zgomot crescător (vedeți figura 28.1).

FIGURA 28.1

Exemplul `SoundView` afișează două buffere și semnalul de undă rezultat din mixarea acestora.



Apoi cele două buffere sunt redade concomitent, la apăsarea de către utilizator a unei taste, producând în bufferul primar o undă sinusoidală care se pierde în zgomotul alb (acest semnal rezultat este afișat în partea inferioară, cu albastru). Frecvența unei sinusoidale este determinată de codul ASCII al tastei apăsate, astfel că literele de la sfârșitul alfabetului vor produce sunete mai înalte.

În mod evident, acest exemplu necesită prezența unei plăci de sunet.

#### LISTINGUL 28.1 LST32\_1.CPP – Un program sintetizator simplu, creat prin mixarea a două buffere `DirectSound` redade de la tastatură

```
1 CSoundView::CSoundView()
2 {
3 m_pDSObject = NULL;
4 m_pDSBuffer = NULL;
5 m_pDSMix = NULL;
6 }
7
8 CSoundView::~CSoundView()
9 {
10 if (m_pDSMix) m_pDSMix->Release();
11 if (m_pDSBuffer) m_pDSBuffer->Release();
12 if (m_pDSObject) m_pDSObject->Release();
13 }
14
```

(continuare)



## LISTINGUL 28.1 Continuare

```

15 void CSoundView::PlayFreq(double dFreq)
16 {
17 if (!m_pDSObject)
18 {
19 // Cream obiectul Direct Sound
20 if (DS_OK==DirectSoundCreate(
21 NULL,&m_pDSObject,NULL))
22 {
23 // Stabilim nivelul de cooperare
24 m_pDSObject->SetCooperativeLevel(m_hWnd,
25 DSSCL_NORMAL);
26
27 // Initializam bufferul de sunet
28 memset(&m_DSBufferDesc,0,sizeof(DSBUFFERDESC));
29
30 // Pregatim un buffer secundar
31 m_DSBufferDesc.dwSize = sizeof(DSBUFFERDESC);
32 m_DSBufferDesc.dwFlags = DSBCAPS_CTRLDEFAULT;
33 m_DSBufferDesc.dwBufferBytes = 4096;
34
35 // Stabilim un format de redare la 22.0Khz, 1 octet
36 // digital-analog
37 static WAVEFORMATEX sWave = {
38 WAVE_FORMAT_PCM,1,22000,22000,1,8,0 };
39
40 m_DSBufferDesc.lpwfxFormat = &sWave;
41
42 // Cream primul buffer secundar
43 m_pDSObject->CreateSoundBuffer(&m_DSBufferDesc,
44 &m_pDSBuffer,NULL);
45
46 // Cream al doilea buffer secundar
47 m_pDSObject->CreateSoundBuffer(&m_DSBufferDesc,
48 &m_pDSMix,NULL);
49 }
50
51 // Daca obiectul DirectSound este valid...
52 if (m_pDSObject)
53 {
54 // Declaram pointeri la buffere
55 LPBYTE pBuffer1,pBuffer2;
56 LPBYTE pEnv1, pEnv2;
57 DWORD dwSize1, dwSize2;
58

```

```

59 // Stergem suprafata ferestrei
60 CClientDC dcClient(this);
61 CRect rcClient;
62 GetClientRect(&rcClient);
63 int yOff = (rcClient.Height()-24)/3;
64 dcClient.FillSolidRect(
65 rcClient,RGB(255,255,255));
66
67 // Blocam bufferul de sunet
68 m_pDSBuffer->Lock(0,m_DSBufferDesc.dwBufferBytes, — 2
69 (LPVOID*)&pBuffer1,&dwSize1,
70 (LPVOID*)&pBuffer2,&dwSize2,0);
71 m_pDSMix->Lock(0,m_DSBufferDesc.dwBufferBytes,
72 (LPVOID*)&pEnv1,&dwSize1,
73 (LPVOID*)&pEnv2,&dwSize2,0);
74
75 // Umplem cele doua buffere
76 double dRange = 128.0 / (double)dwSize1;
77 double dXRange = (double)rcClient.Width()
78 / (double)dwSize1;
79 for(int i=0;i<(int)dwSize1;i++)
80 {
81 double dAmplitude = 1.0 + dRange * (double)i;
82 BYTE b1 = (BYTE)(127 + (128.0 - dAmplitude)
83 * sin((double)i / (0.147 * dFreq)));
84 BYTE b2 = (BYTE)(rand()%((int)dAmplitude)>>1);
85 *(pBuffer1+i) = b1;
86 *(pEnv1+i) = b2;
87
88 BYTE b3 = b1 + b2;
89
90 // Afisam semnalele de unda — 3
91 int x = (int)(dXRange * (double)i);
92 dcClient.SetPixelV(x,
93 (b1>>1),RGB(255,0,0));
94 dcClient.SetPixelV(x,yOff+64+
95 (b2>>1),RGB(0,255,0));
96 dcClient.SetPixelV(x,yOff*2+
97 (b3>>1),RGB(0,0,255));
98 }
99
100 // Deblocam bufferele si redam sunetul — 4
101 m_pDSBuffer->Unlock(pBuffer1,dwSize1,
102 pBuffer2,dwSize2);
103 m_pDSMix->Unlock(pEnv1,dwSize1,
104 pEnv2,dwSize2);

```

(continuare)

## LISTINGUL 28.1 Continuare

```

105 m_pDSBuffer->Play(0,0,0);
106 m_pDSMix->Play(0,0,0);
107 }
108 }
109
110 void CSoundView::OnKeyDown(UINT nChar,UINT nRepCnt,
111 UINT nFlags)
112 {
113 CView::OnKeyDown(nChar, nRepCnt, nFlags);
114
115 // Redam un sunet cu o frecventa egala cu codul tastei
116 if (nRepCnt==1) PlayFreq((double)nChar);
117 }

```

- ❶ Obiectele COM DirectSound trebuie eliberate elegant, prin apelul Release().
- ❷ Funcția Lock() blochează memoria și întoarce pointeri la buffere care vor fi utilizați de program.
- ❸ Semnalul de undă este afișat pe ecran, astfel încât să puteți vizualiza conținutul bufferului.
- ❹ Bufferele sunt redată după deblocarea lor.

Codul prezentat în listingul 28.1 face parte din clasa reprezentare a unei aplicații SDI obișnuite (numită Sound), creată cu AppWizard. Datorită acestor adăugiri în cadrul clasei reprezentare (în fișierul SoundView.cpp), va trebui să adăugați și în fișierul de antet al clasei (în SoundView.h) liniile de mai jos, reprezentând noile variabile membru care sunt folosite în listingul 28.1:

```

IDirectSound* m_pDSObject;
IDirectSoundBuffer* m_pDSBuffer;
IDirectSoundBuffer* m_pDSMix;
DSBUFFERDESC m_DSBufferDesc;
void PlayFreq(double dFreq);

```

Liniile de mai sus declară variabilele membru pointer la interfață necesare pentru DirectSound și pentru cele două buffere, precum și structura de descriere a unui buffer. Tot aici se află și definiția funcției PlayFreq(), folosită pentru implementarea efectelor sonore.

Va trebui să apălați la ClassWizard pentru a adăuga funcția de tratare OnKeyDown() a mesajului WM\_KEYDOWN. În consecință, vor fi generate intrările corespunzătoare în fișierul de antet și în harta de mesaje.

Pentru a informa compilatorul în ceea ce privește interfețele necesare, structura de format al redării și funcția sinus folosită, trebuie să adăugați la începutul fișierului de definiție a clasei (soundview.h) și următoarele linii #include:

```

#include "mmsystem.h"
#include "DSOUND.H"
#include "math.h"

```

Constructorul clasei reprezentare, prezentat în liniile de la 1 la 6 ale listingului 28.1, inițializează pointerul la obiectul DirectSound, m\_pDSObject, și cei doi pointeri la buffere, m\_pDSBuffer și m\_pDSMix, cu valoarea NULL. Funcția destructor corespunzătoare, în liniile de la 8 la 13, eliberează aceste obiecte COM, apelând funcția Release() a interfeței și decrementând astfel contorul de referințe. Liniile de la 17 la 49 ale funcției PlayFreq() inițializează, la primul apel al funcției, obiectul DirectSound necesar și cele două buffere. Inițializarea trebuie să aibă loc aici deoarece apelul SetCooperativeLevel() din linia 24 necesită un indicator de fereastră valid, m\_hWnd, care nu este disponibil la momentul execuției constructorului.

## Nivelurile de cooperare

Obiectul DirectSound permite patru nivele de cooperare. DSSCL\_EXCLUSIVE oferă aplicației un acces exclusiv (dacă este prima care-l solicită) la placa de sunet. Celelalte aplicații vor fi reduse la tăcere. DSSCL\_NORMAL permite cooperarea totală și realizează o excelentă interacțiune multitasking cu celelalte aplicații aflate în execuție. DSSCL\_PRIORITY oferă aplicației dumneavoastră o prioritate față de celelalte aplicații obișnuite la accesarea resurselor plăcii de sunet. DSSCL\_WRITEPRIMARY reprezintă nivelul de prioritate cel mai înalt, oferind aplicației acces la bufferele de sunet primare și blocând redarea oricăror buffere de sunet secundare (ale oricăror aplicații).

Dacă inițializarea din linia 20 a obiectului DirectSound reușește, cele două buffere secundare sunt create prin apelul funcției CreateSoundBuffer(), după pregătirea structurii de descriere a bufferelor, m\_DSBufferDesc, și a structurii swave de tip WAVEFORMATEX.

A doua parte a funcției PlayFreq() umple bufferele de sunet, trasează semnalele de undă și redă sunetele în concordanță cu valorile transmise prin parametrul dFreq. Liniile de la 59 la 65 inițializează un context dispozitiv client, șterg suprafața reprezentării și inițializează un dreptunghi cu coordonatele zonei client. Liniile de la 68 la 73 apelează Lock() pentru a bloca cele două buffere de sunet secundare, furnizând pointeri către acestea. Liniile de la 79 la 98 implementează apoi bucla principală, care generează valorile de undă pentru cele două buffere și le afișează în cadrul reprezentării. Linia 83 calculează valoarea pentru unda sinusoidală, b1, în funcție de frecvență, de poziția din buffer reprezentată de contorul i al buclei și de o valoare descrescătoare dAmplitude a volumului. Această valoare este înscrisă în primul dintre cele două buffere în linia 85 și folosită pentru reprezentarea pe ecran a unei prin apelul funcției SetPixelV() din linia 92, care afișează un punct roșu.

Valoarea aleatoare de zgomot alb pentru al doilea dintre buffere, b2, este determinată apelând funcția rand() pentru obținerea unui număr aleator și calculând jumătate din restul împărțirii acestui număr la intensitatea volumului, dAmplitude, rezultând astfel un nivel crescător de zgomot, ca în linia 84. Această valoare este înscrisă în buffer în linia 86 și afișată pe ecran în linia 94, ca un punct verde. Cea de-a treia valoare, b3, care va fi echivalentă cu conținutul la redare al bufferului primar, se calculează prin simpla mixare (adunare) a celorlalte două valori, b1 + b2, așa cum se vede în linia 88, fiind folosită pentru a afișa semnalul de undă rezultat prin puncte albastre, în linia 96.

**Mixarea bufferelor de sunet**

Prin mixarea semnalelor de undă din două buffere secundare redade simultan, sunetul produs de un semnal de undă poate fi modulat cu celălalt semnal. Această tehnică vă permite să creați într-un buffer un semnal de undă care oscilează cu o frecvență înaltă, modulându-l apoi cu un semnal oscilant de frecvență joasă aflat într-un alt buffer. Astfel se obține un efect de vibrație sau de asprime a sunetului produs.

Liniile de la 101 la 103 apelează `Unlock()` pentru a debloca cele două buffere, pregătindu-le pentru redarea care se va efectua prin apelurile funcției `Play()` pentru fiecare buffer, în liniile 105 și 106. La primul apel al funcției `Play()` este construit automat un buffer primar care reflectă proprietățile bufferului secundar, acesta fiind folosit pentru a trimite datele direct către hardware. Următorul apel `Play()` va duce la mixarea celui de-al doilea buffer secundar cu bufferul primar, rezultând astfel efectul sonor dorit.

În fine, funcția `OnKeyDown()` poate fi adăugată cu ajutorul casetei de dialog `New Windows Message/Event handler`, răspunzând la apăsarea unei taste prin apelul `PlayFreq()`, transmitând codul ASCII al tastei pentru a implementa un sintetizator controlat de la tastatură.

Înainte de a asambla această aplicație, nu uitați să adăugați biblioteca `dsound.lib` în caseta **Object/ Library Modules** din pagina **Link** a casetei de dialog **Project Settings**. Această casetă de dialog poate fi apelată apăsând `Alt+F7` sau efectuând un clic pe meniul **Project** și selectând opțiunea **Settings**.

După asamblarea și rularea aplicației, puteți să apăsați diferite taste pentru a afișa semnalele de undă și a audia efectele sonore produse.

**VEDEȚI...**

➤ Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.

**Utilizarea interfeței DirectDraw**

Interfața `DirectDraw` vă permite să scrieți aplicații de grafică foarte rapide și fără efecte de pâlpare. Această interfață vă oferă posibilitatea de a folosi tehnica de sincronizare a cadrelor prin dublu buffer, necesară pentru jocurile de acțiune și pentru aplicațiile de animație. Puteți să controlați întreaga suprafață de afișare din Windows, să folosiți diferite moduri grafice (inclusiv celebrul `ModeX` folosit, se pare, în jocul `Doom2`) și să comutați suprafața de afișare astfel încât să desenați pe o suprafață ascunsă în timp ce pe ecran este afișată o alta. Această comutare poate fi sincronizată cu momentul în care raza de electroni revine la poziția de început a cadrului, ceea ce permite realizarea unei animații line, fără efecte de pâlpare.

**ModeX**

`ModeX` este un mod VGA care nu a fost documentat. Mulți autori de jocuri au fost obligați să aleagă un mod cu o rezoluție scăzută, dar cu multe culori, pentru a putea atinge o rată de înprospătare suficient de mare pentru realizarea animației. În consecință, cel mai bun candidat a fost modul VGA `0x13` (hexa), care oferă o rezoluție de 320 de pixeli pe orizontală și 200 pe verticală. În schimb, exotical `ModeX` permite o rezoluție ceva mai mare pe verticală, oferind 320x240 pixeli.

Puteți, de asemenea, să păstrați afișarea din Windows și să utilizați facilitățile de desenare rapidă în interiorul unei ferestre obișnuite. `DirectDraw` implementează facilitățile sale extinse de desenare și copiere a blocurilor de imagine prin intermediul a două nivele de software – nivelul de abstractizare hardware (HAL – Hardware Abstraction Layer) și nivelul de emulare hardware (HEL – Hardware Emulation Layer). În momentul în care folosiți funcțiile de trasare și copiere ale unei suprafețe `DirectDraw`, nivelul apelat este HAL.

Este oferit, așadar, un mecanism independent de dispozitiv care permite ca același cod să funcționeze pe o multitudine de plăci grafice. Dacă o funcție de trasare poate fi efectuată de componentele hardware ale plăcii grafice, aceasta va beneficia de imensul spor de viteză adus de implementarea hardware. Dacă nu, HEL va emula funcția în mod transparent, astfel că oricum nu trebuie să vă preocupați de capacitățile hardware ale plăcii grafice.

Interfața `DirectDraw` constă din numai patru obiecte COM principale, accesibile prin intermediul interfețelor, așa cum se vede în tabelul 28.1. Puteți observa că unele dintre interfețe au atașată la nume cifra 2. Motivul este faptul că sunt versiuni ulterioare ale interfețelor COM originale, acestea rămânând disponibile pentru aplicațiile mai vechi. Remarcați, de asemenea, că obiectul utilizat în principal este `DirectDrawSurface`. Acesta este creat de obiectul `DirectDraw` principal, suprafața asociată putând fi decupată cu obiecte `DirectDrawClipper`, iar culorile folosite selectându-se dintr-o paletă `DirectDrawPalette`.

**TABELUL 28.1 Obiectele DirectDraw și funcționalitatea acestora**

| Obiect <code>DirectDraw</code> | Interfață COM                   | Funcționalitate                                                                                                                                                                                                                                                                           |
|--------------------------------|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>DirectDraw</code>        | <code>IDirectDraw2</code>       | Gestionează modurile grafice, caracteristicile dispozitivelor și crearea celorlalte obiecte.                                                                                                                                                                                              |
| <code>DirectDrawSurface</code> | <code>IDirectSurface2</code>    | Gestionează bitmap-urile primare și secundare, tehnica dublu buffer și funcțiile de desenare și de copiere a blocurilor de imagine. Decuparea se face cu ajutorul obiectului <code>DirectDrawClipper</code> , iar culorile pot fi selectate din obiectul <code>DirectDrawPalette</code> . |
| <code>DirectDrawClipper</code> | <code>IDirectDrawClipper</code> | Gestionează lista de informații pentru decupare folosită la decuparea suprafețelor.                                                                                                                                                                                                       |
| <code>DirectDrawPalette</code> | <code>IDirectDrawPalette</code> | Gestionează intrările din paleta de culori și maparea informațiilor pentru culorile folosite în cadrul unei suprafețe.                                                                                                                                                                    |

Obiectul COM `DirectDraw` principal se poate crea cu ajutorul funcției `DirectDrawCreate()` din biblioteca `ddraw.lib` sau prin apelul direct al funcției `CoCreateInstance()`.

Funcția `DirectDrawCreate()` primește trei parametri. Primul este un pointer la un GUID care identifică driverul grafic folosit. În mod normal se transmite valoarea `NULL` pentru a indica utilizarea driverului grafic curent; celelalte drivere pot fi determinate prin enumerare. Al doilea parametru este un pointer care indică un pointer la o interfață `IDirectDraw` pentru noul obiect. Dacă folosiți această metodă, ar trebui să apelați apoi funcția `QueryInterface()` pentru a obține un pointer la mai noua interfață `IDirectDraw2`. Cel de-al treilea parametru este în mod normal `NULL`, permițând altfel agregarea COM.

#### Agregarea COM

Agregarea COM este o tehnică folosită de componentele care conțin alte componente în scopul refolosirii funcționalității acestora. Interfața pe care componenta externă o oferă obiectelor client este de fapt interfața unui obiect intern. În momentul în care un client solicită un pointer la această interfață, componenta externă îi furnizează direct un pointer la interfața obiectului intern care este agregat. Prin intermediul acestui al treilea parametru, obiectul `DirectDraw` permite agregarea sa de către alte obiecte. Versiunile mai vechi ale bibliotecilor DirectX nu oferă această facilități, iar funcția `DirectDrawCreate()` va întoarce un cod de eroare.

Probabil că este mai ușor să creați obiectul COM prin intermediul funcției

`CoCreateInstance()`, deoarece puteți să accesați direct noua interfață `IDirectDraw2` și nu mai este nevoie să stabiliți legătura cu biblioteca `ddraw.lib`. În acest scop, în apelul `CoCreateInstance()` veți transmite `CLSID_DirectDraw` ca identificator de clasă și `IID_IDirectDraw2` pentru a accesa direct noua interfață; acești identificatori sunt definiți în fișierul de antet `DrawDlg.h`. Apoi va trebui să apelați funcția `Initialize()`, transmițându-i valoarea GUID care identifică driverul grafic (de regulă, `NULL`).

Odată ce ați creat un obiect `DirectDraw`, va trebui să stabiliți un nivel de cooperare prin intermediul funcției `IDirectDraw::SetCooperativeLevel()`. Această funcție primește doi parametri; primul este un indicator de fereastră care identifică aplicația posesoare, iar al doilea este o valoare de indicator.

În cazul în care păstrați afișarea normală din Windows, puteți să partajați accesul cu alte aplicații transmițând indicatorul `DDSCCL_NORMAL`, dar dacă doriți să controlați întreaga suprafață de afișare va trebui să transmiteți o combinație între indicatorii `DDSCCL_EXCLUSIVE` și `DDSCCL_FULLSCREEN`. Pentru a folosi tehnica de sincronizare a cadrelor prin dublu buffer, va trebui să preluați controlul asupra întregii suprafețe de afișare.

Crearea unei suprafețe pentru desenare și copiere de imagini se face prin apelul funcției `CreateSurface()` a obiectului `DirectDraw`, transmițând o structură `DDSURFACEDESC` care descrie noua suprafață. Această structură descrie mai multe tipuri de suprafețe. De regulă, veți crea o suprafață principală, specificând indicatorul `DDSCAPS_PRIMARYSURFACE`, și o serie de suprafețe buffer ascunse din care sau în care veți copia blocuri de imagine sau pe care le veți folosi pentru comutarea paginilor (tehnica dublu buffer). Legat de comutarea paginilor, prin apelul `CreateSurface()` pentru crearea bufferului primar puteți să creați automat și să asociați și bufferele secundare, precizând numărul `dwBackBufferCount` al acestora și specificând indicatorul `DDSD_BACKBUFFERCOUNT`.

#### Modificarea dinamică a culorilor pentru realizarea animației

Modificarea dinamică a culorilor poate fi o tehnică utilă pentru obținerea de animație fără costuri. Reducerea volumului de desenare în cadrul unei animații reprezintă întotdeauna un beneficiu major. Prin utilizarea unor palete de culori cu modificare dinamică, puteți să implementați secvențe repetitive de animație simplă în zone de pe ecran care nu necesită redesenare ulterioară. Spre exemplu, dacă ar trebui să desenați cu culori diferite patru cercuri având același centru și diametre crescătoare, ați putea să redefiniți trei culori ca negru și una ca alb. Culoarea albă va fi deplasată în paletă prin cele patru culori pentru fiecare cadru, ducând la colorarea pe rând a fiecărui cerc.

Puteți, de asemenea, să creați mai multe obiecte de decupare care vor defini listele cu informații de decupare. Acestea sunt foarte utile atunci când folosiți `DirectDraw` în afișarea normală din Windows, deoarece puteți să creați liste de decupare dintr-o fereastră părinte pentru a preveni afișarea în exteriorul ferestrei. Aceste obiecte se creează cu ajutorul funcției `CreateClipper()` a obiectului `DirectDraw`, fiind selectate apoi în cadrul unei suprafețe prin apelul funcției `SetClipper()` a suprafeței.

Funcția `CreatePalette()` a obiectului `DirectDraw` permite crearea unor palete cu facilități de modificare dinamică a culorilor; aceste palete pot fi apoi inițializate cu o serie de valori `COLORREF` și selectate în cadrul unei suprafețe prin apelul funcției `SetPalette()` a acesteia.

Desenarea în cadrul suprafeței se face prin intermediul funcțiilor GDI obișnuite, fiind necesară o blocare prealabilă a suprafeței și obținerea unui context dispozitiv prin apelul funcției `GetDC()`. După ce terminați de desenat, va trebui să apelați funcția `ReleaseDC()` a suprafeței. O serie de funcții rapide pentru copierea blocurilor de imagine, un exemplu de funcție fiind `Blit()`, vă permit să copiați și să colorați suprafețele prin intermediul unei game largi de procedee, printre care scalarea și maparea texturilor.

Suprafețele primare și suprafețele buffer ascunse pot fi comutate cu ajutorul funcției `Flip()` a suprafeței primare, aceasta putând să opereze asincron și să aștepte începutul următorului ciclu de înprospătare a imaginii. Alternativ, puteți să specificați un indicator care să determine așteptarea operației de înprospătare sau puteți să apelați funcția `WaitForVerticalBlank()` a obiectului `DirectDraw`. Printre alte funcții utile legate de generarea imaginii se află `GetMonitorFrequency()` și `GetScanLine()`.

#### Utilizarea funcțiilor de copiere a blocurilor de imagine și a celor GDI

Trebuie să aveți grijă să nu apelați o funcție de copiere a blocurilor atunci când există un context dispozitiv valid obținut cu `GetDC()`. `GetDC()` blochează contextul dispozitiv și orice apel al unei astfel de funcții va eșua; înainte de apelarea funcțiilor de copiere a blocurilor trebuie să apelați `ReleaseDC()`.

Listingul 28.2 prezintă un exemplu de aplicație `DirectDraw` care folosește tehnica dublu buffer pentru a crea o spirală hipnotică, animată rapid și cursiv pe întreaga suprafață a ecranului. Nu este nevoie să legați biblioteca `ddraw.lib`, deoarece obiectul `DirectDraw` este creat cu `CoCreateInstance()`; ecranul Windows este refăcut în momentul în care apăsați `Esc`. Exemplul creat este o aplicație de tip dialog, numită `Draw`, în care suprafața de afișare este pegătită în `OnInitDialog()`, iar imaginea este desenată într-un buffer de fundal în cadrul funcției `OnTimer()`, ca răspuns la mesaje generate de Windows la

expirarea unor intervale de timp. Odată ce rapida și complexa operație de desenare se încheie, bufferele primar și de fundal sunt schimbate între ele, astfel că noua imagine este afișată imediat, fără nici o pâlpăire, în timp ce ecranul este inert între două cadre.

**LISTINGUL 28.2 LST32\_2.CPP – DirectDraw este folosit pentru animarea fără pâlpăire a unei spirale hipnotice prin intermediul tehnicii dublu buffer**

```

1 CDrawDlg::CDrawDlg(CWnd* pParent /*=NULL*/)
2 : CDialog(CDrawDlg::IDD, pParent)
3 {
4 //{AFX_DATA_INIT(CDrawDlg)
5 // NOTE: the ClassWizard will add member initializa
6 //}{AFX_DATA_INIT
7 // Note that LoadIcon does not require a subsequent
8 m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
9
10 m_pIDraw = NULL;
11 m_pMainSurface = NULL;
12
13 CoInitialize(NULL); — ❶
14 }
15
16 CDrawDlg::~CDrawDlg()
17 {
18 if (m_pMainSurface)
19 {
20 m_pIDraw->SetCooperativeLevel(m_hWnd, DDSCL_NORMAL);
21 m_pIDraw->RestoreDisplayMode(); — ❷
22 m_pMainSurface->Release();
23 }
24 if (m_pIDraw) m_pIDraw->Release();
25 CoUninitialize();
26 }
27
28 BOOL CDrawDlg::OnInitDialog()
29 {
30 CDialog::OnInitDialog();
31
32 // ** Initializam biblioteca OLE / COM
33 if (FAILED(CoInitialize(NULL))) return FALSE;
34
35 // ** Cream un obiect si o interfata DirectDraw
36 HRESULT hr = CoCreateInstance(CLSID_DirectDraw, NULL, — ❸
37 CLSCTX_ALL, IID_IDirectDraw2, (void**)&m_pIDraw);
38
39 if (!FAILED(hr))
40 {

```

```

41 hr = m_pIDraw->Initialize((struct _GUID*)NULL);
42 if (hr != DD_OK) return FALSE;
43 }
44
45 // ** Selectam modul de afisare exclusiva pe intregul ecran
46 m_pIDraw->SetCooperativeLevel(m_hWnd,
47 DDSCL_EXCLUSIVE|DDSCL_FULLSCREEN
48 | DDSCL_ALLOWREBOOT);
49
50 // ** Pregatim o structura de descriere a suprafetei
51 DDSURFACEDESC DrawSurfaceDesc;
52 memset(&DrawSurfaceDesc, 0, sizeof(DrawSurfaceDesc));
53 DrawSurfaceDesc.dwSize = sizeof(DrawSurfaceDesc);
54 DrawSurfaceDesc.dwFlags = DDSD_CAPS — ❹
55 | DDSD_BACKBUFFERCOUNT;
56 DrawSurfaceDesc.ddsCaps.dwCaps = DDSCAPS_COMPLEX
57 | DDSCAPS_FLIP | DDSCAPS_PRIMARYSURFACE;
58 DrawSurfaceDesc.dwBackBufferCount = 1;
59 // ** Cream suprafata primara
60 hr = m_pIDraw->CreateSurface(&DrawSurfaceDesc,
61 (IDirectDrawSurface**)&m_pMainSurface, NULL);
62 if (FAILED(hr)) return FALSE;
63
64 // ** Determinam bufferul de fundal
65 DrawSurfaceDesc.ddsCaps.dwCaps = DDSCAPS_BACKBUFFER;
66 hr = m_pMainSurface->GetAttachedSurface(— ❺
67 &DrawSurfaceDesc.ddsCaps, &m_pIFlipSurface);
68
69 SetTimer(1, 50, 0); // ** Cream un timer de 50ms
70
71 return TRUE;
72 }
73 void CDrawDlg::OnTimer(UINT nIDEvent)
74 {
75 CDialog::OnTimer(nIDEvent);
76
77 static RECT rc={0,0,0,0};
78 static double dPetals=2.0;
79
80 // ** Stergem repede suprafata, cu o operatie de bloc
81 if (rc.right>0)
82 {
83 DDBLTFX dbltfx;
84 dbltfx.dwSize=sizeof(DDBLTFX);

```

(continuare)

## LISTINGUL 28.2 Continuare

```

85 dbltfx.dwFillColor=RGB(0,0,0);
86 HRESULT hr = m_pIFlipSurface->Blit(&rc,NULL, — 6
87 &rc,DBLT_COLORFILL,&dbltfx);
88 }
89 HDC hdc = NULL;
90
91 // ** Obținem un context dispozitiv pentru suprafața
92 if (m_pIFlipSurface->GetDC(&hdc) == DD_OK)
93 {
94 if (hdc)
95 {
96 CDC dc;
97 dc.Attach(hdc);
98
99 rc.right=dc.GetDeviceCaps(HORZRES);
100 rc.bottom=dc.GetDeviceCaps(VERTRES);
101
102 double mx = (double)(rc.right>>1);
103 double my = (double)(rc.bottom>>1);
104
105 CPen psCol(PS_SOLID,1,RGB(0,255,0));
106 CPen* ppsOld = dc.SelectObject(&psCol);
107
108 double sx = mx/2;
109 double sy = my/2;
110 double sAngle = 0.0;
111
112 // ** Desenăm imaginea animată
113 for(int i=0;i<(int)dPetals;i++)
114 {
115 double s1a = sin(sAngle);
116 double c1a = cos(sAngle);
117 double s2a = sin(sAngle * dPetals);
118 double c2a = cos(sAngle * dPetals);
119 int x = (int)(mx+sx*c1a+sx*c2a);
120 int y = (int)(my+sy*s1a-sy*s2a);
121 if (i==0) dc.MoveTo(x,y);
122 else dc.LineTo(x,y);
123 sAngle+=0.01745;
124 }
125
126 dPetals+=0.1;
127 dc.SelectObject(ppsOld);
128 dc.Detach();
129 }

```

```

130 m_pIFlipSurface->ReleaseDC(hdc);
131 }
132
133 // ** Afisăm imaginea nou desenată
134 m_pIMainSurface->Flip(NULL,DDFLIP_WAIT); — 8
135 }

```

- ❶ CoInitialize() inițializează bibliotecile COM.
- ❷ Modul de afișare revine la NORMAL și obiectele COM sunt eliberate.
- ❸ CoCreateInstance() este folosită pentru a obține un pointer la interfața IDirectDraw2 a unui nou obiect DirectDraw.
- ❹ Structura DrawSurfaceDesc descrie o suprafață primară cu un buffer de fundal.
- ❺ GetAttachedSurface() furnizează un pointer la suprafața de fundal.
- ❻ Funcția Blit() poate fi folosită nu numai pentru copierea de imagini, ci și pentru umplerea unei suprafețe cu o culoare specificată.
- ❼ Aceste relații matematice produc două figuri care se rotesc rapid pentru a crea o spirală în continuă schimbare.
- ❽ Funcția Flip() așteaptă momentul în care ecranul este inert între cadre, după care comută paginile video, evitând astfel pâlpâirea cauzată de paginile afișate parțial.

Pe lângă listingul 28.2, va trebui să asigurați includerea în fișierul de implementare DrawDlg.cpp a unor fișiere care conțin valorile GUID și definițiile de funcții trigonometrice, adăugând următoarele linii #include la început:

```

#include <initguid.h>
#include "DrawDlg.h"
#include "math.h"

```

Definițiile pentru interfețele DirectDraw se pot specifica prin includerea fișierului de antet corespunzător la începutul fișierului DrawDlg.h de definiție a clasei:

```
#include <ddraw.h>
```

Tot în fișierul DrawDlg.h trebuie adăugați și următorii pointeri la interfață, ca variabile membru ale clasei CDrawDlg:

```

-
IDirectDraw2* m_pIDraw;
IDirectDrawSurface2* m_pIMainSurface;
IDirectDrawSurface2* m_pIFlipSurface;

```

În listingul 28.2, în cadrul constructorului CDrawDlg, pointerii la interfață primesc valoarea NULL (în liniile 10 și 11), iar apelul CoInitialize() din linia 13 realizează inițializarea bibliotecii COM.

Funcția OnInitDialog() este folosită pentru crearea obiectului DirectDraw, în linia 36 fiind apelată CoCreateInstance(). Urmează apelul Initialize() din linia 41, care indică utilizarea driverului grafic activ. În continuare, linia 47 determină fixarea nivelului de cooperare exclusiv și afișarea pe întreaga suprafață a ecranului.

Structura `DrawSurfaceDesc`, de tip `DDSURFACEDESC`, este declarată în linia 50 și configurată pentru crearea unui buffer primar cu un singur buffer de fundal pentru tehnica dublu buffer. Această structură va putea fi utilizată apoi pentru crearea suprafețelor de desenare, prin apelul `CreateSurface()` din linia 59; pointerul de interfață la suprafața de fundal suplimentară poate fi obținut cu ajutorul funcției `GetAttachedSurface()`, ca în linia 65.

În fine, apelul `SetTimer()` din linia 68 stabilește un interval de 50ms la expirarea căruia vor fi efectuate operațiile de desenare, implementate în funcția de tratare `OnTimer()` care începe în linia 73. Funcția de tratare `OnTimer()` se adaugă prin intermediul `ClassWizard`, fiind corespunzătoare mesajului `WM_TIMER`. Apelând la `ClassWizard` asigurați crearea corespunzătoare a definițiilor din clasă și a intrărilor din harta de mesaje.

După stabilirea unor valori inițiale, funcția `Blit()` este apelată în linia 86 pentru a acoperi rapid cu negru bufferul de fundal, ștergând orice era desenat acolo.

`GetDC()` este apelată în linia 92 pentru a obține de la GDI un indicator către un context dispozitiv destinat desenării pe suprafața de fundal. Acest indicator poate fi atașat apoi unei clase `CDC`, ca în linia 97. Liniile de la 99 la 126 utilizează apeluri GDI obișnuite și un obiect `CPen` pentru a trasa cu viteză linii pe suprafața care va reprezenta un cadru din animația spiralei. Contextul dispozitiv este eliberat în linia 130, după care bufferele sunt schimbate între ele pentru a afișa cadrul nou desenat, fiind apelată funcția `Flip()` și transmițându-se indicatorul `DDFLIP_WAIT` pentru a aștepta momentul în care ecranul este inert între două cadre.

#### Perioada inertă dintre cadre

La afișarea pe ecranul monitorului (un tub catodic) a datelor furnizate de placa grafică, o rază de electroni parcurge ecranul de la stânga la dreapta, afișând pe rând fiecare linie. Electronii lovesc particulele de fosfor aflate pe suprafața interioară a ecranului, ducând la o emisie momentană de lumină și generând astfel pixelii. În momentul în care raza atinge colțul din dreapta jos al ecranului, ea trebuie să revină în colțul stânga sus fără a mai lumina nici un pixel. Acum începe perioada în care raza de electroni este stinsă, între afișarea a două cadre. Această perioadă este marcată de placa grafică printr-un indicator. Este momentul cel mai potrivit pentru a actualiza conținutul memoriei pe care placa grafică o folosește la generarea imaginilor, deoarece pe monitor nu se afișează nimic.

Funcția `destructor`, dintre liniile 16 și 25, reface suprafața de afișare normală din Windows, resetând nivelul de cooperare și apelând `RestoreDisplayMode()` pentru revenirea la normal a afișării. Interfețele sunt eliberate cu `Release()` și, în mod civilizat, urmează de-inițializarea bibliotecii COM.

Dacă operați aceste modificări în cadrul unei aplicații obișnuite de tip dialog, veți putea să asamblați aplicația fără a mai specifica nici o altă bibliotecă API, ilustrând puterea obiectelor COM. Rulați apoi aplicația și savurați animația rapidă și cursivă, fără pâlpare, care demonstrează adevărata putere oferită de DirectX.

Aplicația poate fi închisă în orice moment prin apăsarea tastei `Esc` (care este o scurtătură pentru butonul `Cancel` încă activ al casetei de dialog). Aceasta va duce la refacerea suprafeței de lucru și închiderea aplicației.

#### VEDEȚI...

- Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.
- În legătură cu contextele dispozitiv și desenarea prin intermediul GDI, vedeți pagina 323.

### Utilizarea interfeței Direct3D

Interfața `Direct3D` vă permite să exploatați puterea plăcilor grafice moderne, care dispun de facilități 3D implementate hardware. Această interfață lucrează împreună cu `DirectDraw` pentru a produce scene tridimensionale realiste, animate cursiv, așa cum puteți vedea în jocuri și în aplicațiile de realitate virtuală.

#### Nolle plăci grafice 3D

De ceva vreme a apărut o nouă gamă de plăci grafice, care implementează o serie de facilități pentru generarea scenelor 3D. Componentele hardware specializate pot efectua volumul mare de calcule complexe de transformare mult mai rapid decât prin software. În plus, procesorul este degrevat de sarcini, astfel încât autorii de jocuri și aplicații CAD au la dispoziție mai mult timp de procesor pentru a îmbunătăți puterea jocului și calitatea imaginii.

`Direct3D` folosește un motor de randare care se compune din trei module, destinate matricilor de transformare a coordonatelor, efectelor de iluminare cu spot luminos și lumină ambientală și generării scenei finale.

Există mai multe interfețe `Direct3D` care oferă toată funcționalitatea necesară pentru construirea și generarea de scene tridimensionale complexe, acestea fiind împărțite în două categorii: directe și elaborate.

Obiectele directe sunt obiecte 3D de bază, de nivel scăzut, care se generează rapid (interfețele corespunzătoare sunt prezentate în tabelul 28.2). Pe aceste obiecte directe sunt construite un număr mare de obiecte elaborate, care dezvoltă funcționalitatea pentru a produce scene și animații 3D sofisticate, de înaltă calitate și în timp real.

TABELUL 28.2 Interfețele `Direct3D` și funcționalitățile acestora

| Interfața <code>Direct3D</code>     | Funcționalitate                                                                                                                                                                                                    |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>IDirect3D</code>              | Inițializează mediul și creează obiectele <code>Light</code> , <code>Material</code> și <code>Viewport</code> .                                                                                                    |
| <code>IDirect3DDevice</code>        | Gestionează configurarea caracteristicilor 3D hardware și a caracteristicilor mediului. Această interfață se poate obține pe baza unei suprafețe <code>DirectDraw</code> , apelând <code>QueryInterface()</code> . |
| <code>IDirect3DExecuteBuffer</code> | Bufferele de execuție sunt folosite pentru a stoca informații despre vertexuri și instrucțiuni de randare pentru elemente specifice ale unei scene 3D gata de randare.                                             |
| <code>IDirect3DLight</code>         | Configurează parametrii diferitelor surse de lumină dintr-o scenă.                                                                                                                                                 |
| <code>IDirect3DMaterial</code>      | Administrează proprietățile de material și parametrii de reflectanță sau transparență.                                                                                                                             |

(continuare)



TABELUL 28.2 Interfețele Direct3D și funcționalitățile acestora

| Interfața Direct3D | Funcționalitate                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IDirect3DTexture   | Administrează bitmap-uri de textură pentru suprapunerea peste suprafețe 3D.                                                                                                                        |
| IDirect3DViewport  | Grupează efectele de iluminare, textură și fundal dintr-o poziție de pe ecran. Acestea sunt îmbinate apoi pentru a permite desenarea în cadrul dispozitivului, împreună cu informațiile de vertex. |

## Utilizarea interfeței DirectPlay

Interfața DirectPlay permite implementarea facilă a unui joc desfășurat în rețea, între mai mulți parteneri, fără a mai necesita efortul de implementare a suportului pentru diferite tipuri de conexiuni posibile. Cele două interfețe, IDirectPlay2 și IDirectPlayLobby, gestionează toate aspectele privind obiectele unui joc cu mai mulți participanți și, respectiv, administrarea sesiunilor.

### Suportul pentru rețea oferit de DirectPlay

Interfața DirectPlay permite conectarea calculatoarelor prin cablu serial, rețele locale TCP/IP, legături telefonice și Internet, Netware și prin multe alte tipuri de legături de rețea.

## Utilizarea interfeței DirectInput

Interfața DirectInput oferă un acces mai rapid la mouse și tastatură decât cel permis de interfața Windows. În plus, permite utilizarea dispozitivelor joystick multi-axă cu mai multe butoane, dispunând de funcții pentru calibrarea acestora. Această interfață este simplă, dar flexibilă, fiind implementată prin intermediul a două interfețe COM. Interfața IDirectInput gestionează dispozitivele de intrare instalate în sistem și starea acestora. Vă permite, de asemenea, să creați instanțe ale interfeței IDirectInputDevice pentru a gestiona configurarea și operarea dispozitivelor individuale.

## Utilizarea interfeței DirectSetup

Interfața DirectSetup este probabil cea mai mică interfață pentru programarea aplicațiilor. Ea conține numai două funcții: DirectXSetup(), care automatizează instalarea și configurarea tuturor componentelor DirectX, și DirectXRegisterApplication(), care înregistrează un joc ca aplicație de rețea cu utilizatori multipli, operând prin DirectPlayLobby.

## Crearea de mesaje și poștă electronică prin intermediul MAPI

Interfața pentru programarea aplicațiilor de mesagerie de la Microsoft (MAPI –Messaging API) nu este de fapt un pachet API; este o serie de standarde care compun o arhitectură. Urmând aceste standarde rigide și complexe, producătorii de software pot să separe și să creeze diferitele componente ale unui sistem de poștă electronică.

Majoritatea producătorilor vor dori, probabil, să creeze aplicații client proprii (pentru citire și scriere), în timp ce alții implementează furnizorii de servicii care răspund de transportul și depozitarea mesajelor sau de gestionarea destinatarilor printr-o agendă de adrese. Fiecare dintre aceste componente poate fi implementată separat, iar utilizatorii pot să îmbine și să combine soluțiile pentru a crea sisteme de mesagerie integrate și compatibile. Această flexibilitate este necesară mai ales în organizațiile mari, care trebuie să integreze unitar sisteme de poștă electronică vechi și noi, foarte diferite între ele.

Atunci când scrieți o aplicație MAPI, puteți opta între două abordări, în funcție de complexitatea operațiilor de mesagerie necesare. Interfața MAPI de bază include un minim de funcții necesare pentru deschiderea unei sesiuni de lucru MAPI, expedierea sau citirea mesajelor și rezolvarea (găsirea celei mai bune potriviri) adreselor pe baza unei agende de adrese. Dacă aceste facilități fundamentale sunt suficiente pentru aplicația dumneavoastră, este de preferat să folosiți interfața MAPI de bază. Orice altă funcționalitate necesită interfața MAPI extinsă, implementată prin fișierul mapi28.dll și componentele furnizoare de servicii pentru care optați. Interfața MAPI extinsă este, în mod necesar, imensă și complexă; pentru a o utiliza este nevoie de o bună înțelegere a tehnologiei COM, deoarece este implementată în esență printr-o ierarhie obiectuală de obiecte și interfețe COM.

### Pachetul SDK pentru interfața MAPI extinsă și programul exemplificativ MDBView

Dacă aveți într-adevăr nevoie de suportul oferit de interfața MAPI extinsă, pregătiți-vă pentru multă muncă și învățare. Va trebui să descărcați pachetul SDK pentru MAPI. Acest pachet conține un excelent program exemplificativ, MDBView.exe, împreună cu codul său sursă, care vă poate ajuta considerabil în a înțelege complexitatea programării cu interfața MAPI extinsă.

### VEDEȚI...

➤ Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.

## Utilizarea interfeței MAPI de bază

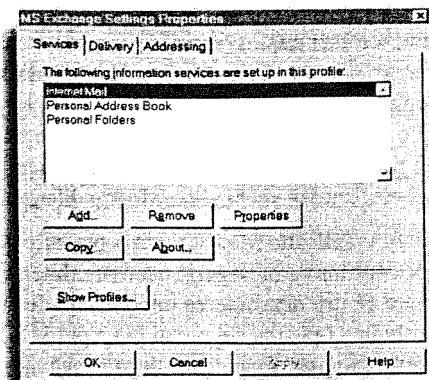
Interfața MAPI de bază este implementată în fișierul mapi28.dll, iar prototipurile de funcții și valorile indicator sunt definite în fișierul de antet mapi.h. Deschiderea unei sesiuni MAPI de bază și conectarea la un furnizor de servicii se face specificând un profil MAPI sau acceptând profilul implicit. Profilurile MAPI pot fi configurate prin intermediul miniaplicației Mail and Fax din panoul de control, ilustrată în figura 28.2. Fiecare profil conține informații despre furnizorii de transport, de mesaje și de agende de adrese pe care un program client îi va utiliza la deschiderea unei sesiuni, așa cum sunt Microsoft Exchange sau Lotus Notes. Odată deschisă o sesiune, puteți să expediați și să recepționați mesaje e-mail, închizând în final sesiunea de lucru.

### Miniaplicația Mail and Fax din panoul de control

Miniaplicația Mail and Fax din panoul de control poate avea un aspect diferit sau poate să figureze sub un alt nume, în funcție de aplicațiile e-mail MAPI care sunt instalate curent în sistem.

FIGURA 28.2

Miniaplicația Mail and Fax din panoul de control permite configurarea profilurilor MAPI.



Funcțiile MAPI de bază sunt prezentate în tabelul 28.3. Aceste funcții pot fi apelate dintr-o aplicație fie prin legarea directă cu biblioteca `mapi28.lib`, care oferă punctele de intrare exportate pentru `mapi28.dll`, fie apelând funcțiile `LoadLibrary()` și `GetProcAddress()` pentru a încărca și descărca dinamic biblioteca DLL.

TABELUL 28.3 Funcțiile MAPI de bază

| Funcție                          | Descriere                                                                            |
|----------------------------------|--------------------------------------------------------------------------------------|
| <code>MAPILogon()</code>         | Deschide o sesiune de lucru MAPI prin intermediul unui profil de configurare         |
| <code>MAPILogoff()</code>        | Închide o sesiune de lucru MAPI                                                      |
| <code>MAPISendMail()</code>      | Expediază un mesaj pe baza unei structuri <code>MapiMessage</code>                   |
| <code>MAPISendDocuments()</code> | Expediază o mulțime de fișiere document prin specificarea numelor acestora           |
| <code>MAPIFindNext()</code>      | Localizează primul sau următorul mesaj și întoarce un identificator al acestuia      |
| <code>MAPIReadMail()</code>      | Citește informațiile unui mesaj într-o structură <code>MapiMessage</code>            |
| <code>MAPISaveMail()</code>      | Salvează noul mesaj într-o casuță poștală, dacă furnizorul de servicii o permite     |
| <code>MAPIDeleteMail()</code>    | Șterge mesajul specificat de ultimul apel <code>MAPIFindNext()</code>                |
| <code>MAPIFreeBuffer()</code>    | Eliberează bufferele alocate de MAPI                                                 |
| <code>MAPIAddress()</code>       | Afișează caseta de dialog cu lista de adrese, permițând editarea de către utilizator |
| <code>MAPIDetails()</code>       | Furnizează componentele unei anumite adrese                                          |
| <code>MAPIResolveName()</code>   | Transformă numele aproximativ al unui destinatar într-o adresă precisă               |

### Funcțiile MAPI de bază și extinse

Fișierul `MAPI32.DLL` conține codul de implementare atât pentru funcțiile MAPI de bază, cât și pentru cele extinse.

### VEDEȚI...

► Pentru a înțelege ideile și conceptele legate de OLE și COM, vedeți pagina 581.

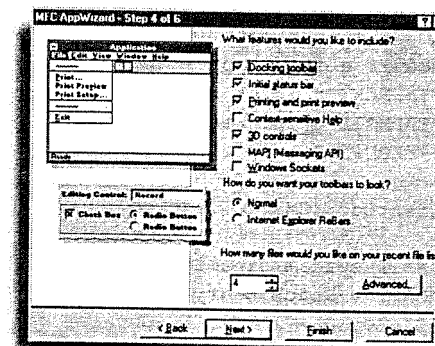
## Utilizarea AppWizard pentru adăugarea facilității de expediere a mesajelor prin intermediul MAPI

Pasul 4 din AppWizard vă permite să adăugați într-o infrastructură SDI sau MDI suport pentru MAPI (vedeți figura 28.3), selectând caseta de validare **MAPI (Messaging API)** din secțiunea **What features would you like to include?**. În consecință, meniul **File** al noii aplicații va conține o opțiune **Send**, iar în clasa document vor fi adăugate următoarele intrări în harta de mesaje:

```
ON_COMMAND(ID_FILE_SEND_MAIL, OnFileSendMail)
ON_UPDATE_COMMAND_UI(ID_FILE_SEND_MAIL,
 OnUpdateFileSendMail)
```

FIGURA 28.3

Adăugarea de suport pentru MAPI într-o infrastructură SDI sau MDI prin intermediul AppWizard.



Aplicația generată poate fi asamblată și rulată fără nici o modificare, oferindu-vă deja posibilitatea de a expedia e-mail. Puteți să efectuați un clic pe meniul **File** și să selectați opțiunea **Send** pentru a apela caseta de dialog pentru deschiderea unei sesiuni MAPI. După ce selectați un profil (sau configurați sistemul de poștă electronică), aplicația de e-mail va fi lansată și veți putea să compuneți și să expediați un mesaj. Infrastructura va serializa automat documentul curent, apelând `OnSaveDocument()`, și-l va atașa la noul mesaj.

Dacă aveți nevoie de facilități e-mail mai sofisticate în cadrul aplicației, va trebui să apelați la funcțiile MAPI de bază sau la cele extinse.

Listingul 28.3 ilustrează utilizarea interfeței MAPI de bază într-o infrastructură SDI cu suport MAPI și reprezentare de tip editor, rezultatul fiind un client de poștă care poate atât să expedieze mesaje, prin opțiunea **Send** existentă în meniul **File**, cât și să citească

mesajele, prin opțiunea **Receive** adăugată în meniul **File**. Mesajele sunt afișate în reprezentarea de tip editor.

Exemplul prezentat încarcă biblioteca mapi28.dll dinamic, în cadrul constructorului documentului, și deschide o nouă sesiune MAPI, iar la final, în funcția destructor, închide sesiunea și descarcă biblioteca DLL.

Creați o infrastructură SDI obișnuită cu AppWizard și selectați suportul pentru MAPI, asigurând astfel adăugarea opțiunii **Send**. În ultimul pas din AppWizard veți opta pentru o reprezentare de tip editor. Apoi va trebui să apelați la editorul de resurse pentru a adăuga în meniul **File** o nouă opțiune de meniu, numită **Receive**. Codul necesar se adaugă într-o funcție de tratare pentru noua opțiune de meniu, în cadrul clasei document.

### LISTINGUL 28.3 LST32\_3.CPP – Utilizarea interfeței MAPI de bază pentru citirea mesajelor e-mail într-o aplicație SDI cu suport MAPI

```
1 #include <mapi.h>
2
3 HMODULE g_hMAPI;
4 LHANDLE g_hSession;
5
6 LPMAPILOGON g_lpfnLogon;
7 LPMAPILOGOFF g_lpfnLogoff;
8 LPMAPIFINDNEXT g_lpfnFindNext;
9 LPMAPIREADMAIL g_lpfnReadMail;
10 LPMAPIFREEBUFFER g_lpfnFreeBuffer;
11
12 CMailClientDoc::CMailClientDoc()
13 {
14 // ** Incarcam dinamic biblioteca DLL si functiile necesare
15 g_hMAPI = LoadLibrary("MAPI28.DLL");
16 g_lpfnLogon = (LPMAPILOGON)
17 GetProcAddress(g_hMAPI, "MAPILogon"); — 1
18 g_lpfnLogoff = (LPMAPILOGOFF)
19 GetProcAddress(g_hMAPI, "MAPILogoff");
20 g_lpfnFindNext = (LPMAPIFINDNEXT)
21 GetProcAddress(g_hMAPI, "MAPIFindNext");
22 g_lpfnReadMail = (LPMAPIREADMAIL)
23 GetProcAddress(g_hMAPI, "MAPIReadMail");
24 g_lpfnFreeBuffer = (LPMAPIFREEBUFFER)
25 GetProcAddress(g_hMAPI, "MAPIFreeBuffer");
26
27 (*g_lpfnLogon)(0, NULL, NULL, — 2
28 MAPI_NEW_SESSION | MAPI_LOGON_UI, 0, &g_hSession);
29 }
30
31 CMailClientDoc::~CMailClientDoc()
```

```
32 {
33 // ** Inchidem sesiunea si descarcam biblioteca
34 (*g_lpfnLogoff)(g_hSession, 0, 0, 0);
35 FreeLibrary(g_hMAPI);
36 }
37
38 void CMailClientDoc::OnFileReceive()
39 {
40 // ** Accesam reprezentarea
41 POSITION pos = GetFirstViewPosition();
42 CView* pView = GetNextView(pos);
43
44 // ** Declaram identificatorii de mesaj
45 static char szSeedMessage[512];
46 static char szMessage[512];
47
48 static LPSTR pSeed = NULL;
49 static LPSTR pMsg = szMessage;
50 lpMapiMessage *lpMessage = NULL;
51
52 // ** Cautam urmatorul mesaj
53 ULONG ulResult = (*g_lpfnFindNext)(g_hSession, — 3
54 (ULONG)pView->m_hWnd, NULL, pSeed,
55 MAPI_LONG_MSGID, 0L, pMsg);
56
57 // ** Citim urmatorul mesaj
58 ulResult = (*g_lpfnReadMail)(g_hSession,
59 (ULONG)pView->m_hWnd, pMsg, MAPI_PEEK, 0L, &lpMessage); — 4
60
61 CString strMessage;
62 CString strFmt;
63 if (ulResult == 0L)
64 {
65 // ** Stocam componentele mesajului
66 strFmt.Format("From: %s\r\n",
67 lpMessage->lpOriginator->lpszName);
68 strMessage += strFmt;
69 strFmt.Format("To: %s\r\n",
70 lpMessage->lpRecips[0].lpszName);
71 strMessage += strFmt;
72 strFmt.Format("Subject: %s\r\n",
73 lpMessage->lpszSubject);
74 strMessage += strFmt;
75 strMessage += lpMessage->lpszNoteText;
76 pSeed = szSeedMessage;
77 strcpy(pSeed, pMsg);
78 }
```

(continuare)

## LISTINGUL 28.3 Continuare

```

79 // ** Inscriem mesajul in reprezentarea de tip editor
80 pView->SetWindowText(strMessage);
81
82 // ** Eliberam bufferul alocat
83 (*g_lpfnFreeBuffer)((LPVOID)lpMessage);
84 }
85 }

```

- ❶ Funcțiile DLL sunt accesate manual, prin intermediul funcției `GetProcAddress()`.
- ❷ Programul deschide o sesiune de lucru MAPI folosind profilul implicit.
- ❸ `FindNext()` caută următorul e-mail pornind de la cel anterior.
- ❹ `ReadMail()` citește conținutul mesajului într-un buffer `MapiMessage`.

Prima linie din listingul 28.3 asigură includerea fișierului de antet `map.h`, care conține toate prototipurile și valorile de indicator folosite. Funcția constructor a documentului, aflată între liniile 12 și 29, încarcă biblioteca `map28.dll` prin apelul `LoadLibrary()` din linia 15, după care determină adresele funcțiilor din această bibliotecă, apelând `GetProcAddress()` și înscrind adresele întoarse în pointerii globali declarați între liniile 6 și 10. În fine, linia 27 deschide o nouă sesiune de lucru MAPI, reținând indicatorul de sesiune în `g_hSession`.

Verificarea valorii întoarse de `GetProcAddress()`

După încărcarea cu succes a bibliotecii DLL, funcțiile acesteia pot fi obținute apelând `GetProcAddress()`. Funcția `GetProcAddress()` întoarce adresa funcției al cărei nume este transmis ca parametru, fiind necesară și transmiterea unui indicator către biblioteca DLL.

Dacă `GetProcAddress()` a reușit să găsească adresa dorită, pointerul de funcție `lpfnLogon` va conține această adresă; altfel, acest pointer va avea valoarea `NULL`. Verificați întotdeauna dacă `GetProcAddress()` întoarce un pointer de funcție valid. Nu există modalitate mai sigură de a bloca programul decât să determinați execuția de la adresa zero!

Funcția de tratare `OnFileReceive()`, care începe cu linia 38, trebuie adăugată în clasa document cu `ClassWizard`, aici fiind implementată citirea mesajelor e-mail. Liniile 41 și 42 determină reprezentarea de tip editor, folosită în linia 53 ca indicator de fereastră părinte la identificarea următorului mesaj existent, prin apelul `FindNext()`. Bufferul `szSeedMessage` oferă un indiciu funcției `FindNext()`, permițându-i să determine următorul mesaj disponibil pe baza conținutului mesajului curent, primul mesaj fiind furnizat atunci când pointerul `pSeed` este `NULL`.

În linia 58 se încearcă citirea mesajului identificat de pointerul `lpMessage` printr-un apel `ReadMail()`. Dacă apelul reușește, în `ulResult` este înscrisă valoarea zero, iar pointerul `lpMessage` indică o structură de tip `MapiMessage`. Componentele mesajului pot fi extrase apoi din această structură, prin intermediul pointerului `lpMessage` (între liniile 65 și 75).

Odată ce șirul `strMessage` a fost formatat pe baza informațiilor din structura de mesaj, puteți să afișați acest șir în cadrul reprezentării cu ajutorul funcției `SetWindowText()`, ca în linia 80, eliberând apoi structura printr-un apel al funcției `MAPIFreeBuffer()`, efectuat prin intermediul pointerului de funcție `g_lpfnFreeBuffer` (pregătit în linia 24), așa cum o arată linia 83.

Liniile 76 și 77 copiază mesajul curent în bufferul pregătit ca indiciu pentru următorul apel `FindNext()`, permițând extragerea următorului mesaj în momentul în care utilizatorul selectează iarăși noua opțiune de meniu **Receive**.

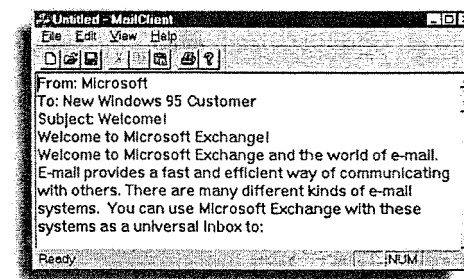
La închiderea documentului, sesiunea de lucru este închisă în linia 34, iar biblioteca este descărcată prin apelul `FreeLibrary()` din linia 35.

Dacă asamblați și rulați această aplicație, veți avea posibilitatea de a trimite și primi mesaje e-mail, așa cum o ilustrează figura 28.4, efectuând clic pe meniul **File** și selectând **Send** sau noua opțiune **Receive**. Mesajele pot fi citite pe rând, selectând de fiecare dată opțiunea **Receive**.

La prima rulare a aplicației va fi afișată o casetă de dialog pentru deschiderea unei sesiuni MAPI, fiind afișat profilul implicit; un simplu clic pe **OK** va deschide sesiunea de lucru pe baza profilului selectat.

FIGURA 28.4

Citirea mesajelor e-mail într-o aplicație SDI cu o reprezentare de tip editor, prin intermediul interfeței MAPI de bază.



## Utilizarea bibliotecilor multimedia video și audio

Facilitățile multimedia au schimbat imaginea de cal de povară obosit al mediului de afaceri pe care o aveau calculatoarele personale. Calculatoarele pot acum să redea compact-discuri, programe de televiziune, efecte sonore și filme, toate aceste facilități putând fi adăugate în propriile aplicații grație bibliotecii multimedia și a interfeței pentru controlul multimedia (MCI – Media Control Interface). MCI și mecanismul asociat pentru interfața cu utilizatorul – fereastra `MCIWnd` – oferă o modalitate simplă și unitară pentru controlul unor elemente software și hardware foarte sofisticate.

## Utilizarea interfeței pentru controlul multimedia

În scopul controlării diverselor dispozitive multimedia din cadrul aplicațiilor, MCI vă permite să optați între un sistem de comandă bazat pe text și unul bazat pe structuri.

Sistemul de comandă bazat pe text (Command Strings – șiruri de comandă) vă permite să compuneți o secvență de mai multe comenzi de control, transmițând-o apoi către MCI prin intermediul funcției `mciSendString()`. Acest sistem este simplu și ușor de folosit, fiind de ajutor prin depanarea unor șiruri inteligibile.

#### Comenzile MCI

Comenzile MCI sunt concepute pentru a fi cât mai generice posibil, ceea ce înseamnă existența unui nucleu de comenzi care pot fi utilizate pentru orice dispozitiv. Aceste comenzi de bază vă permit operații de închidere, deschidere, determinare a stării și a caracteristicilor, încărcare de date, salvare de date, redare, înregistrare, întrerupere, reluare a redării sau a înregistrării, oprire sau accesare a unei poziții arbitrare, oricare ar fi tipul dispozitivului multimedia.

Ca alternativă, sistemul de comandă bazat pe structuri (Command Messages – mesaje de comandă) vă permite să exprimați comenzile sub formă de structuri și indicatori, trimise apoi către MCI prin intermediul funcției `mciSendCommand()`. Acest sistem s-ar putea să fie ceva mai rapid, deoarece nu mai este necesară compunerea șirurilor și apoi analiza lor. Oricum, cele două sisteme sunt identice din punct de vedere funcțional, iar dumneavoastră puteți să-l alegeți pe acela care se potrivește cel mai bine în contextul aplicației.

Prima comandă trimisă către MCI trebuie să determine deschiderea unui anumit dispozitiv. În acest scop, fie veți apela `mciSendString()` specificând șirul `open`, fie veți apela `mciSendCommand()` specificând indicatorul `MCI_OPEN`. Puteți deschide oricare dintre dispozitivele prezentate în tabelul 28.4, cu condiția ca respectivul dispozitiv să fie instalat în sistem. Unele dispozitive necesită hardware specializat, în timp ce pentru altele, printre care `waveaudio` și `digitalvideo`, există software de redare inclus în mod standard în Windows.

**TABELUL 28.4 Tipuri de dispozitive MCI**

| Șirul asociat dispozitivului | Indicatorul de mesaj asociat dispozitivului | Descriere                                       |
|------------------------------|---------------------------------------------|-------------------------------------------------|
| waveaudio                    | MCI_DEVTYPE_WAVEFORM_AUDIO                  | Redă/înregistrează fișiere .wav de semnal audio |
| sequencer                    | MCI_DEVTYPE_SEQUENCER                       | Secvențiator MIDI                               |
| cdaudio                      | MCI_DEVTYPE_CD_AUDIO                        | Redă compact-discuri                            |
| dat                          | MCI_DEVTYPE_DAT                             | Redă/înregistrează casete audio digitale        |
| digitalvideo                 | MCI_DEVTYPE_DIGITAL_VIDEO                   | Redă fișiere .avi de video digital              |
| vcr                          | MCI_DEVTYPE_VCR                             | Redă/înregistrează casete video                 |
| videodisc                    | MCI_DEVTYPE_VIDEODISC                       | Redă video-discuri                              |
| mmovie                       | MCI_DEVTYPE_ANIMATION                       | Redă animație                                   |

| Șirul asociat dispozitivului | Indicatorul de mesaj asociat dispozitivului | Descriere                          |
|------------------------------|---------------------------------------------|------------------------------------|
| overlay                      | MCI_DEVTYPE_OVERLAY                         | Intrare video analogă în fereastră |
| scanner                      | MCI_DEVTYPE_SCANNER                         | Scanează imagini                   |
| other                        | MCI_DEVTYPE_OTHER                           | Dispozitiv indefinit               |

După cum probabil vă așteptați, comanda `open` necesită o serie de parametri, astfel că într-un apel `mciSendString()` va trebui să transmiteți un șir în care `open` este urmat de parametrii de mai jos, separați prin spații:

```
open nume_dispozitiv indicatori_deschidere indicatori_notificare
```

Parametrul `nume_dispozitiv` poate fi oricare dintre șirurile de dispozitiv enumerate în tabelul 28.4. Parametrul `indicatori_deschidere` poate să specifice un pseudonim (care identifică în comenzi ulterioare instanța particulară a dispozitivului deschis), precum și diferite opțiuni specifice dispozitivului. Parametrul `indicatori_notificare` poate fi `wait` sau `notify`, determinând programul să aștepte deschiderea dispozitivului sau să continue și să fie notificat în momentul în care deschiderea s-a efectuat.

#### Numele dispozitivelor MCI

Interfața MCI este extensibilă, fiind posibilă crearea de noi dispozitive MCI prin adăugarea și înregistrarea driverelor de dispozitiv corespunzătoare. În Windows 95, intrările de înregistrare se găsesc în fișierul `C:\Windows\System.ini`, în secțiunea `[mci]`. În Windows NT, aceste intrări se găsesc în cadrul cheii `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SYS...MCI`.

De asemenea, puteți să determinați funcția `mciSendString()` să furnizeze un mesaj de stare, transmițând un buffer pentru mesaj și dimensiunea acestuia. Dacă doriți notificarea programului, puteți să transmiteți și indicatorul asociat ferestrei care va primi mesajul corespunzător.

Spre exemplu, următorul șir de comandă transmis către MCI prin apelul `mciSendString()` deschide dispozitivul de redare a compact-discurilor și îi asociază numele `mycd` pentru apelurile ulterioare:

```
char szRetMsg[80];
mciSendString("open cdaudio alias mycd wait",
 &szRetMsg, sizeof(szRetMsg), m_hWnd);
```

Această comandă va aștepta până când dispozitivul de redare a CD-urilor este deschis corespunzător, întorcând "1" în bufferul `szRetMsg` dacă totul merge bine; în caz contrar, deschiderea a eșuat și în bufferul `szRetMsg` este înscris "0". În cazul în care operația de deschidere eșuează, este întoarsă o eroare `MCIERROR`; aceasta poate fi convertită într-un mesaj de eroare prin intermediul funcției `mciGetErrorString()`.

Funcția `mciSendCommand()` poate să efectueze aceeași operație de deschidere a dispozitivului de redare a CD-urilor, fiind necesară transmiterea a patru parametri:

- Un identificator de dispozitiv (nefolosit la deschidere)
- Un indicator de comandă (MCI\_OPEN)
- Un indicator de notificare – MCI\_WAIT sau MCI\_NOTIFY
- Un pointer la o structură MCI\_OPEN\_PARMS

#### Caracteristicile dispozitivelor MCI

Verificarea caracteristicilor unui dispozitiv MCI se face prin intermediul comenzii capabilities. Spre exemplu, pentru a verifica dacă un CD are capacitate de înregistrare, puteți formula comanda capabilities cdaudio can record, testând valoarea întoarsă. Indicatorul de comandă echivalent este MCI\_GETDEVCAPS.

Funcția va întoarce un cod de stare MCIERROR, asemeni funcției mciSendString().

Așadar, pentru deschiderea dispozitivului de redare a CD-urilor puteți să formulați următorul mesaj de comandă MCI:

```
MCI_OPEN_PARMS myCDOpen = {NULL, 0, "cdaudio", NULL, NULL};
errOpen = mciSendCommand(NULL, MCI_OPEN,
 MCI_OPEN_TYPE|MCI_WAIT, (DWORD)&myCDOpen);
```

Observați că în apelul mciSendCommand() este transmisă structura myCDOpen, de tip MCI\_OPEN\_PARMS. Această structură este definită astfel:

```
typedef struct {
 DWORD dwCallback;
 MCIDEVICEID wDeviceID;
 LPCSTR lpstrDeviceType;
 LPCSTR lpstrElementName;
 LPCSTR lpstrAlias;
} MCI_OPEN_PARMS;
```

Structura MCI\_OPEN\_PARMS vă permite să transmiteți prin dwCallback un indicator asociat ferestrei care va fi notificată. Identificatorul noului dispozitiv deschis va fi întors în wDeviceID, putând fi utilizat în apeluri ulterioare în scopul identificării dispozitivului. Membrul lpstrElementName se folosește pentru specificarea unui nume de fișier, în cazul în care dispozitivul este deschis cu un fișier asociat, așa cum ar fi un fișier .wav de semnal audio sau un fișier .avi de video digital. Este posibilă precizarea unui pseudonim în lpstrAlias, dar acesta este mai puțin necesar atunci când utilizați mesaje de comandă, deoarece dispozitivul poate fi identificat prin wDeviceID.

Există o mulțime de structuri care conțin diverșii parametri pentru mesajele de comandă MCI\_ corespunzătoare, tot atâtea câte comenzi pot fi transmise, împreună cu parametri, prin șiruri de comandă.

După deschiderea unui dispozitiv, puteți transmite comenzi de redare, înregistrare, închidere, precum și multe alte comenzi multimedia pentru controlul dispozitivului deschis. O parte dintre mesajele și șirurile de comandă existente este prezentată în tabelul

28.5. Pentru fiecare comandă există mai mulți parametri posibili – mult prea mulți pentru a-i discuta aici, dar sper că v-ați format deja o idee.

TABELUL 28.5 Exemple de mesaje și șiruri de comandă

| Mesaj de comandă | Șir de comandă | Descriere                            |
|------------------|----------------|--------------------------------------|
| MCI_OPEN         | open           | Deschide un dispozitiv               |
| MCI_CLOSE        | close          | Închide dispozitivul                 |
| MCI_PLAY         | play           | Redă prin intermediul dispozitivului |
| MCI_RECORD       | record         | Înregistrează de la dispozitiv       |
| MCI_STOP         | stop           | Oprește redarea sau înregistrarea    |
| MCI_SEEK         | seek           | Accesează o poziție arbitrară        |
| MCI_PAUSE        | pause          | Întrerupe dispozitivul               |
| MCI_RESUME       | resume         | Reia după întrerupere                |
| MCI_LOAD         | load           | Încarcă un conținut                  |
| MCI_SAVE         | save           | Salvează conținutul                  |
| MCI_STATUS       | status         | Obține informații de stare           |
| MCI_WINDOW       | window         | Afișează o fereastră video           |
| MCI_WHERE        | where          | Poziționează afișajul                |

### Mesajele de notificare MCI

Atunci când specificați indicatorii MCI\_WAIT sau wait într-o comandă MCI, programul va aștepta execuția comenzii. Această opțiune poate fi utilă în unele situații, dar în cazul în care comandați redarea unui CD aplicația va aștepta până când redarea se încheie, iar o oră ar putea fi un timp de așteptare destul de lung pentru un utilizator! În consecință, puteți să nu mai transmiteți nici un indicator de așteptare sau notificare, iar comanda va reveni imediat și va permite execuția în continuare a programului. Există și posibilitatea de a transmite un indicator MCI\_NOTIFY, astfel încât comanda play să notifice aplicația în momentul în care s-a încheiat redarea. Aplicația ar putea atunci să afișeze un mesaj, să deschidă unitatea de CD sau să efectueze o altă operație. Majoritatea comenzilor MCI permit notificarea.

#### Folosirea procedurilor de întrerupere

Dacă specificați indicatorul de așteptare, ați putea dori ca MCI să apeleze periodic o funcție care să efectueze anumite operații sau verificări asupra stării MCI pe parcursul așteptării. Aceasta se face cu ajutorul funcției mciSetYieldProc(). Funcția primește un identificator de dispozitiv ca prim parametru, adresa funcției cu apel invers ca al doilea parametru și o valoare DWORD conținând date specifice care vor fi transmise funcției cu apel invers. În cazul în care acest apel reușește, este întoarsă valoarea TRUE. Funcția reciprocă mciGetYieldProc() poate fi utilizată pentru a obține adresa procedurii de întrerupere curentă pentru un dispozitiv anume.

Dacă specificați MCI\_NOTIFY sau notify, comanda va reveni imediat, generând la încheiere un mesaj de notificare ce va fi prelucrat de bucla de mesaje a ferestrei al cărei indicator l-ați transmis. Această fereastră ar putea fi o reprezentare, un dialog sau orice altă

clasă derivată din `CWnd` și care este atașată la o fereastră validă (puteți transmite membrul `m_hWnd` al oricărui obiect `CWnd` valid).

Adăugarea într-o aplicație MFC a unei funcții de tratare pentru mesajul `MM_MCINOTIFY` se face prin specificarea unei macrodefiniții `ON_MESSAGE` în harta de mesaje a ferestrei vizate, așa cum o ilustrează liniile de mai jos:

```
BEGIN_MESSAGE_MAP(CMcidlg, CDialog)
 ON_MESSAGE(MM_MCINOTIFY, OnMCINotify)
END_MESSAGE_MAP()
```

Ați putea, în continuare, să implementați o funcție de tratare care primește doi parametri, reprezentând identificatorul dispozitivului care a transmis mesajul de notificare și codul de stare furnizat de comandă. Această funcție de tratare ar trebui să arate astfel:

```
void CMcidlg::OnMCINotify(WPARAM wFlags, LPARAM lDevID)
{
 AfxMessageBox("MCI Notify: Your Command has finished!");
}
```

Eventualul parametru `wFlags` poate să furnizeze funcției de tratare unul dintre codurile de stare prezentate în tabelul 28.6. `lDevID` identifică dispozitivul care a trimis notificarea. Dacă ați folosit șiruri de comandă și pseudonime de identificare a dispozitivelor, pentru a obține identificatorul asociat unui dispozitiv puteți să apelați funcția `mciGetDeviceID()`, transmițându-i ca unic parametru pseudonimul dispozitivului.

**TABELUL 28.6 Coduri de stare posibile într-un mesaj de notificare**

| Cod de stare                       | Descriere                                                                                                                                                                                           |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>MCI_NOTIFY_SUCCEEDED</code>  | Comanda s-a încheiat cu succes.                                                                                                                                                                     |
| <code>MCI_NOTIFY_FAILURE</code>    | A apărut o eroare la efectuarea comenzii.                                                                                                                                                           |
| <code>MCI_NOTIFY_ABORTED</code>    | Notificarea pentru o comandă precedentă a fost anulată de o comandă ulterioară transmisă aceluiași dispozitiv și care este incompatibilă cu prima comandă.                                          |
| <code>MCI_NOTIFY_SUPERSEDED</code> | O nouă comandă a solicitat notificare din partea aceluiași dispozitiv căruia i-a fost transmisă o comandă anterioară. Cele două comenzi sunt compatibile și comanda anterioară este acum înlocuită. |

#### Verificarea valorii întoarse de funcția de comandă

Verificați întotdeauna valoarea întoarsă de funcția de comandă care solicită notificarea. În cazul în care această funcție întoarce imediat un cod de eroare care indică eșecul, mesajul de notificare nu va mai fi transmis niciodată. Aceasta poate duce la situații în care programul se blochează în așteptarea unui mesaj de notificare care nu mai sosește niciodată.

Listingul 28.4 ilustrează utilizarea MCI prin ambele forme – șiruri de comandă și mesaje de comandă – pentru a reda 5 secunde dintr-un compact-disc, de la secunda 12 a pistei 3 până la secunda 17 a aceleiași piste. Acest cod poate fi adăugat oriunde într-o aplicație

MFC (funcția `InitDialog()` a unei aplicații de tip dialog este o soluție rapidă și ușoară pentru acest experiment).

Pentru a defini indicatorii și prototipurile MCI, trebuie să adăugați o linie care să includă fișierul de antet corespunzător:

```
#include "mmsystem.h"
```

Aplicațiile care folosesc biblioteca MCI ar trebui să fie legate cu fișierul bibliotecă `winmm.lib`. Acesta trebuie specificat în caseta **Object/Library Modules** din pagina **Link** a casetei de dialog **Project Settings**.

#### LISTINGUL 28.4 LST32\_4.CPP – Redarea cu MCI a cinci secunde dintr-un compact-disc, prin șiruri de comandă și prin mesaje de comandă

```
1 // Redam 5 secunde din pista 3 a CD-ului folosind șiruri de comanda
2 char szRetMsg[80];
3 char szErrorMessage[512];
4 MCIERROR errOpen;
5 errOpen = mciSendString("open cdaudio alias mycd wait", 1
6 szRetMsg, sizeof(szRetMsg), m_hWnd);
7 if (szRetMsg[0] != '1')
8 {
9 // Anuntam eroare la deschidere
10 mciGetErrorString(errOpen, szErrorMessage, 512);
11 AfxMessageBox(szErrorMessage);
12 }
13 else
14 {
15 // fixam formatul la pista/minut/secunda/cadru
16 mciSendString("set mycd time format tmsf",
17 szRetMsg, sizeof(szRetMsg), m_hWnd);
18 // Redam 5 secunde incepand cu secunda 12 a pistei 3
19 mciSendString(
20 "play mycd from 3:0:12:0 to 3:0:17:0 wait",
21 szRetMsg, sizeof(szRetMsg), m_hWnd);
22 // ** Inchidem dispozitivul
23 mciSendString("close mycd",
24 szRetMsg, sizeof(szRetMsg), m_hWnd);
25 }
26
27 // Redam 5 secunde din pista 3 a CD-ului folosind mesaje de comanda
28 MCI_OPEN_PARMS myCDOpen={NULL, 0, "cdaudio", NULL, NULL}; 2
29 errOpen = mciSendCommand(NULL, MCI_OPEN,
30 MCI_OPEN_TYPE|MCI_WAIT, (DWORD)&myCDOpen);
31 if (errOpen)
32 {
```

(continuare)



## LISTINGUL 28.4 Continuare

```

33 // Anuntam eroare la deschidere
34 mciGetErrorString(errOpen,szErrorMessage,512);
35 AfxMessageBox(szErrorMessage);
36 }
37 else
38 {
39 // fixam formatul la pista/minut/secunda/cadru
40 MCI_SET_PARMS setParams = {NULL,
41 MCI_FORMAT_TMSF,0};
42 errOpen=mciSendCommand(myCDOpen,wDeviceID,MCI_SET,
43 MCI_SET_TIME_FORMAT,(DWORD)&setParams);
44
45 // Redam 5 secunde incepand cu secunda 12 a pistei 3
46 MCI_PLAY_PARMS playParams = {NULL,
47 MCI_MAKE_TMSF(3, 0, 12, 0),
48 MCI_MAKE_TMSF(3, 0, 17, 0)};
49 mciSendCommand(myCDOpen.wDeviceID,MCI_PLAY,
50 MCI_FROM | MCI_TO | MCI_WAIT,
51 (DWORD)&playParams);
52 // ** Inchidem dispozitivul
53 MCI_GENERIC_PARMS genParams = {NULL};
54 mciSendCommand(myCDOpen.wDeviceID,MCI_CLOSE,
55 NULL,(DWORD)&genParams);
56 }

```

- ❶ Șirul de comandă open stabilește un pseudonim pentru referințe ulterioare.
- ❷ Indicatorul MCI\_OPEN este echivalentul într-un mesaj de comandă al șirului open.
- ❸ Limitele intervalului de redare sunt stabilite într-un mesaj de comandă cu ajutorul macrodefiniției MCI\_MAKE\_TMSF.

În linia 5 a listingului 28.4, apelul funcției `mciSendString()` deschide dispozitivul de redare a CD-urilor, stabilind pseudonimul `mycd`. Dacă acest apel eșuează, liniile 10 și 11 afișează mesajul de eroare corespunzător, obținut prin intermediul funcției `mciGetErrorString()` pe baza codului de eroare.

Linia 16 stabilește formatul de timp (time format) la `tmsf`, indicând faptul că orice comenzi `position` ulterioare trebuie interpretate ca specificând pista, minutul, secunda și cadrul.

Linia 19 începe redarea CD-ului prin intermediul comenzii `play`, specificând comenzile opționale `from` și `to` pentru a reda pista 3 între secunde 12 și 17. Indicatorul `wait` face ca funcția să revină doar după redarea fragmentului precizat. Linia 23 închide apoi dispozitivul.

Același fragment este redat cu ajutorul mesajelor de comandă, dispozitivul fiind deschis din nou prin transmiterea mesajului `MCI_OPEN` în linia 29. Formatul de timp este stabilit

printr-un mesaj `MCI_SET`, specificându-se indicatorul `MCI_SET_TIME_FORMAT`, de alegerea formatului `tmsf` fiind responsabil membrul `MCI_FORMAT_TMSF` al structurii `setParams`.

Linia 49 are ca efect redarea fragmentului de mai devreme, aici fiind utilizată macrodefiniția `MCI_MAKE_TMSF` pentru a defini în cadrul structurii `MCI_PLAY_PARMS` limitele intervalului de redare. În fine, linia 54 închide dispozitivul prin transmiterea unui mesaj `MCI_CLOSE`.

## Adăugarea unei ferestre MCI

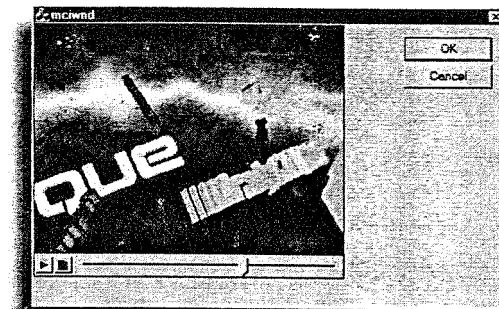
Secțiunea anterioară s-a ocupat de folosirea MCI exclusiv prin cod, fie fără nici o interfață cu utilizatorul, fie cu o interfață proprie; există, însă, o interfață standard de nivel mai înalt pe care o puteți folosi: `MCIWnd`.

`MCIWnd` este un control de nivel înalt care vă oferă controlul total asupra unui dispozitiv MCI. Interfața prezentată de acest control diferă de la un dispozitiv la altul. Pentru un dispozitiv `digitalvideo` veți vedea controale de redare video și o fereastră de imagine; în schimb, pentru dispozitivul `waveaudio` veți vedea doar butoanele de redare, derulare și întrerupere, relevante pentru redarea fișierelor `.wav`.

Adăugarea controlului `MCIWnd` în cadrul propriei aplicații se face incredibil de ușor. De exemplu, simpla adăugare a unui apel de funcție la sfârșitul funcției `InitDialog()` are ca efect afișarea în caseta de dialog a unui control de redare `digitalvideo (.avi)` complet funcțional, așa cum se vede în figura 28.5.

FIGURA 28.5

fereastră MCI atașată la o casetă de dialog standard într-un simplu apel de funcție.



Funcția `MCIWndCreate()` creează un control `MCIWnd` inițializat cu fișierul `.avi` de redat:

```

HWND hMCI = MCIWndCreate(m_hWnd, AfxGetApp()->m_hInstance,
CIWNDF_SHOWALL, "C:\\Vidclip.avi");

```

Primul parametru este `m_hWnd`, un indicator către fereastra părinte (casetă de dialog). Al doilea parametru este un indicator către instanța aplicației. Cel de-al treilea parametru vă permite să specificați o serie întreagă de indicatori care controlează crearea și stilul ferestrei, așa cum o arată tabelul 28.7. Ultimul parametru vă permite să precizați un nume de fișier cu o extensie recunoscută de MCI sau un dispozitiv MCI care să fie deschis. Tipul controlului afișat va corespunde tipului de fișier sau de dispozitiv specificat.

Dacă operația de deschidere reușește, în `hMCI` este întors un indicator valid către fereastra MCI; altfel, `hMCI` capătă valoarea `NULL`.

**TABELUL 28.7 Indicatorii de creare pentru `MCIWndCreate`**

| Indicator                             | Descriere                                                                                                                          |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>MCIWNDF_SHOWALL</code>          | Sunt folosiți toți indicatorii <code>SHOW</code> .                                                                                 |
| <code>MCIWNDF_SHOWNAME</code>         | Afișează numele dispozitivului sau al fișierului în bara de titlu a ferestrei MCI.                                                 |
| <code>MCIWNDF_SHOWMODE</code>         | Afișează în bara de titlu starea dispozitivului, care ar putea fi în redare, oprit, nepregătit, în căutare, deschis sau întrerupt. |
| <code>MCIWNDF_SHOWPOS</code>          | Afișează în bara de titlu poziția curentă de redare.                                                                               |
| <code>MCIWNDF_RECORD</code>           | Afișează un buton de înregistrare în cazul în care dispozitivul are capacitatea de a înregistra.                                   |
| <code>MCIWNDF_NOAUTOSIZEWINDOW</code> | Fereastra nu este dimensionată automat dacă dimensiunea imaginii se modifică.                                                      |
| <code>MCIWNDF_NOAUTOSIZEMOVIE</code>  | Împiedică dimensionarea imaginii în funcție de dimensiunile ferestrei atunci când acestea se modifică.                             |
| <code>MCIWNDF_NOERRORDLG</code>       | Previne afișarea mesajelor de eroare MCI.                                                                                          |
| <code>MCIWNDF_NOMENU</code>           | Înlătură butonul pentru meniu.                                                                                                     |
| <code>MCIWNDF_NOOPEN</code>           | Împiedică utilizatorul să deschidă fișiere.                                                                                        |
| <code>MCIWNDF_NOPLAYBAR</code>        | Previne afișarea unei interfețe cu utilizatorul – manipularea se poate face doar prin cod.                                         |
| <code>MCIWNDF_NOTIFYALL</code>        | Transmite ferestrei părinte toate mesajele care urmează.                                                                           |
| <code>MCIWNDF_NOTIFYMODE</code>       | Transmite ferestrei părinte mesajele de schimbare a stării.                                                                        |
| <code>MCIWNDF_NOTIFYPOS</code>        | Transmite ferestrei părinte mesajele de modificare a poziției de redare.                                                           |
| <code>MCIWNDF_NOTIFYMEDIA</code>      | Transmite ferestrei părinte mesajele de schimbare a fișierului sau a dispozitivului MCI.                                           |
| <code>MCIWNDF_NOTIFYSIZE</code>       | Transmite ferestrei părinte mesajele de modificare a dimensiunilor ferestrei MCI.                                                  |
| <code>MCIWNDF_NOTIFYERROR</code>      | Transmite ferestrei părinte mesajele de eroare MCI.                                                                                |

#### Modificarea stilurilor ferestrei MCI

Stilurile unei ferestre MCI existente pot fi modificate prin intermediul funcției `MCIWndChangeStyles()`. Această funcție primește un indicator către fereastra MCI, o mască de biți care indică stilurile ce pot fi schimbate și o valoare cu noile opțiuni de stil. O macrodefiniție reciprocă, `MCIWndGetStyles()`, întoarce opțiunile de stil curente pentru o fereastră specificată.

Odată ce fereastra MCI este deschisă, îi puteți transmite mesaje de control prin intermediul unei serii de macrodefiniții. Acestea trimit mesaje precum `MCI_PLAY` și multe altele,

corespunzătoare comenzilor MCI pe care le-am prezentat în tabelul 28.5. Există multe macrodefiniții pentru transmiterea de mesaje către `MCIWnd`; tabelul 28.8 înfățișează câteva exemple care corespund comenzilor prezentate în tabelul 28.5.

**TABELUL 28.8 Macrodefiniții pentru mesaje `MCIWnd`**

| MCIWind Macro                 | MCI Message                                           | Descriere                                                                                         |
|-------------------------------|-------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| <code>MCIWndOpenDialog</code> | <code>MCIWNDM_OPEN</code>                             | Deschide un nou dispozitiv sau fișier MCI.                                                        |
| <code>MCIWndOpen</code>       | <code>-1</code>                                       | Deschide un nou dispozitiv sau fișier MCI.                                                        |
| <code>MCIWndClose</code>      | <code>MCI_CLOSE</code>                                | Închide dispozitivul sau fișierul MCI deschis.                                                    |
| <code>MCIWndPlay</code>       | <code>MCI_PLAY</code>                                 | Redă dispozitivul sau fișierul deschis.                                                           |
| <code>MCIWndRecord</code>     | <code>MCI_RECORD</code>                               | Înregistrează din dispozitivul deschis.                                                           |
| <code>MCIWndStop</code>       | <code>MCI_STOP</code>                                 | Oprește redarea sau înregistrarea.                                                                |
| <code>MCIWndSeek</code>       | <code>MCI_SEEK</code>                                 | Efectuează poziționarea dispozitivului.                                                           |
| <code>MCIWndPause</code>      | <code>MCI_PAUSE</code>                                | Întrerupe dispozitivul.                                                                           |
| <code>MCIWndResume</code>     | <code>MCI_RESUME</code>                               | Reia în urma întreruperii.                                                                        |
| <code>MCIWndPutDest</code>    | <code>MCIWNDM_PUT_DEST</code>                         | Poziționează fereastra de afișare.                                                                |
| <code>MCIWndPlayFromTo</code> | <code>MCI_SEEK,</code><br><code>MCIWNDM_PLAYTO</code> | Redă de la o poziție la alta, trimițând mai întâi un mesaj de poziționare și apoi unul de redare. |

#### Determinarea stării curente a ferestrei

După emiterea unei comenzi către o fereastră MCI, ați putea dori să știți care este starea curentă a acesteia (în redare, oprită, etc.). Macrodefiniția `MCIWndGetMode()` vă permite să determinați starea ferestrei specificate. Trebuie să transmiteți, de asemenea, un pointer la un buffer și dimensiunea acestui buffer, aici fiind înscris un șir sau o structură care descrie starea curentă. Stările posibile sunt `MCIMODE_NOT_READY`, `MCI_MODE_OPEN`, `MCI_MODE_PLAY`, `MCI_MODE_RECORD`, `MCI_MODE_PAUSE`, `MCI_MODE_SEEK` și `MCI_MODE_STOP`.

La apelul acestor macrodefiniții trebuie să transmiteți indicatorul către fereastra MCI, fiind posibilă în unele cazuri specificarea unor parametri adiționali. Spre exemplu, pentru a reda un fișier .avi între pozițiile 1 și 5 ați putea să folosiți macrodefiniția de mai jos, presupunând că apelul `MCIWndCreate()` de mai devreme a furnizat un indicator de fereastră `hMCI` valid:

```
MCIWndPlayFromTo(hMCI, 1, 5);
```

Fereastra poate fi distrusă la fel de ușor precum a fost creată, transmițând indicatorul de fereastră macrodefiniției `MCIWndDestroy`:

```
MCIWndDestroy(hMCI);
```

Ceea ce face de fapt această macrodefiniție este să trimită ferestrei mesajul cunoscut `WM_CLOSE` pentru închidere.

Puteți să încercați aceste macrodefiniții `MCiWnd` în orice aplicație care poate pune la dispoziție un indicator către o fereastră părinte validă.

Pentru declarațiile de macrodefiniții și de prototipuri de funcții va trebui să includeți fișierul de antet `Vfw.h`, ca aici:

```
#include "vfw.h"
```

Utilizarea funcțiilor API `MCiWnd` necesită legarea la aplicație a bibliotecii `vfw28.lib`, specificând numele acesteia în caseta **Object/Library Modules** din pagina **Link** a casetei de dialog **Project Settings**. Această casetă de dialog se accesează efectuând un clic pe meniul **Project** și selectând opțiunea **Settings**.

#### VEDEȚI...

➤ În legătură cu utilizarea `MCiWnd` ca și control `ActiveX`, vedeți pagina 189.

## Glosar

**acces** Vedeți *acces privat*, *protejat* sau *public*.

**acces privat** Un specificator pentru declarațiile din cadrul unei clase, care împiedică accesarea directă a variabilelor sau funcțiilor membru de către alte clase fără utilizarea unor funcții de acces sau a unor construcții friend.

**acces protejat** Un specificator pentru declarațiile din cadrul unei clase, care împiedică accesarea directă a variabilelor sau funcțiilor membru de către alte clase fără utilizarea unor funcții de acces sau a unor construcții friend, permițând totuși accesul dintr-o clasă derivată.

**acces public** Un specificator pentru declarațiile din cadrul unei clase, care permite accesarea directă a variabilelor sau funcțiilor membru de către orice alte clase.

**activ OLE** Un obiect sau reprezentare care are capacitatea de a utiliza legarea și încapsularea obiectelor în scopul manipulării, afișării și partajării elementelor de date din alte programe.

**ActiveX** Vedeți *legarea și încapsularea obiectelor*.

**adresă de bază** O adresă care reprezintă poziția de început a unui obiect în memorie.

**adresă de deplasament** Adresă folosită împreună cu o adresă de bază pentru a identifica un membru al unei anumite instanțe de obiect.

**adresă URL** Adresele URL reprezintă un mod codificat de specificare a locațiilor diferitelor tipuri de resurse, în Internet sau pe discurile locale. De exemplu,

<http://www.chaos1.demon.co.uk/> arată că la adresa de Internet [www.chaos1.demon.co.uk](http://www.chaos1.demon.co.uk/) se află o resursă de tip hipertext (`http`).

**alinie** Modul în care textul este afișat relativ la o poziție fixă, de regulă la stânga, la dreapta sau centrat în poziția respectivă.

**ancorare** Conectarea unei ferestre bară de controale, așa cum este o bară cu instrumente, o bară de dialog sau o bară de stare, la o fereastră cadru, astfel încât bara de controale să apară ca făcând parte din însăși fereastra cadru.

**anizotrop** Cu proprietăți diferite în direcții diferite.

**ANSI** American National Standards Institute (Institutul Național de Standarde din America) – În programare, un set de caractere în care fiecare caracter este reprezentat pe un octet.

**API** Vedeți *interfață de programare a aplicațiilor*.

**AppWizard** Instrument de dezvoltare care permite crearea unui proiect Visual C++.

**arhitectură deschisă de servicii Windows** O colecție de standarde, precum TCP/IP Sockets, RPC sau MAPI, folosite în scopul integrării sistemelor Windows împreună cu alte medii de rețea.

**arhivă** Fișier care conține toate datele necesare unei aplicații sub forma unui șir continuu de elemente de date.

**asertiune** Un test folosit la depanarea codului cu scopul de a avertiza programatorul dacă o condiție specificată într-o macrodefiniție `ASSERT` nu are

valoarea TRUE. De obicei, instrucțiunile ASSERT sunt folosite pentru a asigura corectitudinea datelor transmise unei funcții.

**asincron** Execuția programului continuă fără a se aștepta răspunsul unui dispozitiv care este ocupat cu efectuarea unei anumite operații (de exemplu, așteptarea perioadei dintre afișarea a două cadre).

**ASSERT** O instrucțiune care verifică adevărul unei condiții. Macrodefinițiile de aserțiune sunt folosite la depanarea programelor.

**bară de derulare** Control orientat orizontal sau vertical, care conține o casetă mobilă ce poate fi deplasată de-a lungul său pentru a reprezenta o poziție variabilă.

**bază de ancorare** Specifică locul în care o bară de controale poate fi ancorată de o fereastră cadru.

**bibliotecă cu legare dinamică** O bibliotecă ce conține cod obiect sau resurse compilate și care poate fi legată cu un executabil la momentul execuției; aflată de regulă sub forma unui fișier .dll.

**bibliotecă cu legare statică** O bibliotecă conținând cod obiect compilat și care poate fi legată de o aplicație la momentul compilării, aflându-se de regulă sub forma unui fișier bibliotecă .lib.

**biblioteca de clase de bază Microsoft** O colecție de clase livrată împreună cu compilatorul Visual C++, cu scopul de a ajuta și a oferi asistență în dezvoltarea rapidă a aplicațiilor.

**bibliotecă de import** Bibliotecă cu legare statică ce conține doar codul care identifică și apelează funcțiile dintr-o bibliotecă DLL, fiind de regulă generată automat de către compilator.

**biblioteca MFC** Vedeți *biblioteca de clase de bază Microsoft*.

**bit blitting** Copierea rapidă a unor blocuri rectangulare de pixeli dintr-o zonă de memorie în alta – de regulă cu ajutorul unor circuite hardware.

**bitmap** Un vector de date care reprezintă o imagine în culori.

**bitmap de fundal** O imagine afișată ca fundal pentru interfața cu utilizatorul a aplicației.

**browser** O aplicație sau o reprezentare care afișează datele într-o anumită formă. Spre exemplu, Microsoft Internet Explorer 4 este un browser de Web care afișează paginile de hipertext din World Wide Web pe baza limbajului HTML.

**buffer** Un vector de octeți folosit ca zonă de stocare temporară la transferul datelor înspre și dinspre un fișier de pe disc sau la efectuarea operațiilor de intrare/ieșire.

**buffer pentru semnal de undă** Zonă de memorie care conține valorile variabile ale amplitudinii de pe durata unei unde audio sau a unui clip audio.

**butoane de opțiune** Un grup de elemente din care la un moment dat nu poate fi selectat decât unul singur. În momentul selectării unui element, toate celelalte sunt deselectate automat.

**cadru elastic** Selectarea unui element, a mai multora sau a unei regiuni prin apăsarea unui buton al mouse-ului, tragerea unui dreptunghi elastic pentru a defini zona de selecție și apoi eliberarea butonului mouse-ului. Folosit de regulă pentru selectarea mai multor elemente grafice.

**calibrare** Măsurarea unui echipament în condiții de test astfel încât la funcționarea în condiții reale să poată fi asigurată

corectitudinea unei scale pentru echipamentul respectiv.

**caracter** Privește un simbol alfabetic sau numeric definit în cadrul unui font.

**caracteristicile dispozitivului** Specificațiile tehnice ale unui dispozitiv, atașate la un context dispozitiv, care influențează diferitele efecte și funcții de desenare ce pot fi aplicate.

**casetă de mesaj modală** O casetă de mesaj care așteaptă răspunsul utilizatorului înainte de a permite continuarea programului.

**celulă** Privește suprafața rectangulară folosită pentru afișarea unui caracter dintr-un font, inclusiv spațiul.

**clasă de interfață** Vedeți *clasă dispecer*.

**clasă dispecer** O clasă care oferă funcții de interfațare ce pot fi apelate pentru a invoca metode ale unei interfețe COM, executând codul care implementează aceste metode în cadrul unui obiect distinct.

**clase colecție** Clase care gestionează stocarea și manipularea unei mulțimi de obiecte, așa cum sunt vectorii și listele înlanțuite.

**Clipboard** Un concept din Windows reprezentând un buffer comun pentru schimburi de date. Utilizatorii pot să decupeze sau să copieze informațiile în Clipboard, acestea putând fi apoi lipite în alte programe care rulează.

**CLSID** Un identificator al unei clase COM, un număr global unic pe 128 de biți care identifică fiecare clasă de obiecte COM.

**cod de implementare** Secțiune de cod care conține corpul funcțiilor membru ale unei clase.

**coduri de taste virtuale** Valori independente de dispozitiv care definesc apăsările de taste (de pildă, VK\_ESCAPE).

**coduri de tipărire** Numere bine definite care se transmit unei imprimante pentru a solicita o funcție anume, cum ar fi derularea foii curente și încărcarea foii următoare.

**COM/OLE** Vedeți *modelul componentelor obiectuale* și *legarea și încapsularea obiectelor*.

**conectivitate deschisă a bazelor de date** O colecție de drivere standard care permite aplicațiilor să folosească funcții generice pentru a accesa mai multe baze de date diferite.

**constructor** O funcție specială a unei clase C++ care are același nume cu clasa și este folosită pentru inițializarea cu un set de parametri la declararea unui obiect sau la crearea prin intermediul operatorului new.

**container** În terminologia OLE, o aplicație care poate să conțină și să afișeze alte aplicații într-o zonă încadrată. De regulă se integrează cu un obiect încapsulat printr-o serie de funcții standard.

**context de modul** Un pointer la informații care descriu atributele unice și contextul de memorie ale unui anumit obiect, așa cum ar fi un program .exe aflat în execuție sau o bibliotecă DLL legată dinamic.

**context dispozitiv** Un obiect din Windows care prezintă o interfață standard între interfața cu dispozitivele grafice și dispozitivele de reprezentare bidimensionale, așa cum sunt ecranele și imprimantele.

**controale ActiveX** Controale care pot fi partajate între programe, dispunând de proprietăți configurabile ce pot fi stabilite sau determinate prin interfețe OLE/COM.

Aceste controale pot fi utilizate în paginile World Wide Web, descărcate și folosite dinamic, sau pot fi integrate în aplicații pentru a accelera procesul de dezvoltare.

**control de editare dinamic** Control de editare care este creat pentru utilizare temporară. Implică, în mod normal, alocarea de memorie pentru control, utilizarea sa și apoi ștergerea imediată.

**control tri-stare** O casetă de validare sau un buton de opțiune care suportă o a treia stare intermediară sau nedeterminată, reprezentată de regulă prin afișarea voalată a controlului.

**coordonată** O mulțime de numere care reprezintă o poziție precisă prin distanțele relative de-a lungul fiecărei dimensiuni într-un sistem de referință. Coordonatele bidimensionale conțin distanțele de-a lungul unor axe x și y.

**coordonate logice** La utilizarea modurilor de mapare, sistemul de coordonate poate fi configurat astfel încât să reprezinte coordonate reale, nu de tip pixel. Acestea se numesc coordonate logice și unitățile de măsură în care se exprimă depind de modul particular de mapare.

**CScrollView** Clasă MFC oferind funcționalitatea de bază pentru reprezentări care trebuie să înfățișeze informații pe o suprafață mai mare decât cea a ferestrei. Această funcționalitate este implementată prin utilizarea unor bare de derulare orizontală și verticală, atașate la reprezentare pentru a indica poziția ferestrei curente în cadrul întregii suprafețe a reprezentării.

**culoare de desenare** La editarea imaginilor, precizează culoarea cu care se va desena atunci când este apăsat butonul stâng al mouse-ului.

**culoare de ecran** Folosită la editarea pictogramelor pentru a crea transparență.

**culoare de fundal** La editarea imaginilor, precizează culoarea cu care se va desena atunci când este apăsat butonul drept al mouse-ului.

**culoare inversă** Folosită la editarea unei pictograme pentru a specifica părțile din imagine care vor fi inversate la tragerea pictogramei respective.

**cursor** Resursă de tip imagine care reprezintă cursorul mouse-ului. Un cursor are un punct senzitiv care furnizează coordonatele exacte ale mouse-ului.

**DAO (obiecte de acces la bazele de date)** Un sistem specific Microsoft de accesare a bazelor de date, destinat anume bazelor de date care folosesc motorul Microsoft Jet.

**DDE** *Vedeți schimb dinamic de date.*

**a decupa** Secționarea unui desen astfel încât afișarea să aibă loc numai în interiorul unei regiuni specificate. Similar cu desenarea prin șablon.

**decupare** *Vedeți funcții de decupare.*

**definiție de clasă** Fragment de cod, aflat de regulă într-un fișier de antet (.h), care definește funcțiile și variabilele membru care compun o clasă. Definițiile încep cu cuvântul cheie `class` din C++.

**depanare** Operația de examinare a motivelor pentru care un program nu rulează așa cum se dorește, urmată de găsirea unei soluții și corectarea problemei.

**destructor** O funcție specială a unei clase C++ care are același nume cu clasa, prefixat însă de caracterul ~ (tildă). Destructorul poate să elibereze memoria alocată pe durata de viață a clasei și să implementeze orice alt cod de curățare necesar clasei.

**dimensionare fixă** Fontul este afișat astfel încât toate caracterele au aceeași lățime. 'I' și 'l' ocupă tot atâta spațiu cât 'W' și 'M'. Astfel de fonturi sunt folosite adesea la afișarea de numere formate.

**dimensionare proporționată** Fontul este afișat astfel încât caracterele mai subțiri, precum 'I' și 'l', ocupă mai puțin spațiu decât caracterele mai late, precum 'W' și 'M'.

**dimensiune** O măsură a cantității care nu se află în relație matematică directă cu nici o altă cantitate. De regulă descrie lățimea și înălțimea.

**dinamic** În timp ce programul rulează, pe parcurs.

**.dll** Bibliotecă cu legare dinamică conținând cod executabil care implementează un control ActiveX sau funcții disponibile pentru un program .exe sau pictograme și alte resurse compilate.

**DLL** *Vedeți bibliotecă cu legare dinamică.*

**document** O colecție de date.

**Document/Reprezentare** O paradigmă de programare. *Vedeți, de asemenea, interfață document singular și interfață document multiplu.*

**dublu clic** Acțiune specială efectuată de utilizator prin operarea unei succesiuni rapide de două clicuri asupra unui buton al mouse-ului. Este folosit, de obicei, ca scurtătură pentru selectarea unui element dintr-o listă.

**editare imediată** Editarea unei etichete direct în cadrul unui control listă sau arbore. Spre exemplu, redenumirea unui fișier în Windows Explorer se face prin editare imediată.

**eveniment** Mesaj transmis de Windows aplicației pentru a vă informa că utilizatorul a interacționat cu aceasta într-un anume fel sau că ceva s-a modificat.

**.exe** Cod executabil care reprezintă punctul de pornire pentru programul unei aplicații.

**fereastră multi-secțiune dinamică** O fereastră multi-secțiune ale cărei secțiuni sunt create și înlăturate dinamic, în urma acțiunilor utilizatorilor.

**fereastră multi-secțiune statică** Fereastră multi-secțiune ale cărei secțiuni sunt create la utilizare și nu pot fi eliminate de către utilizator.

**format** Standard convenit pentru reprezentarea datelor.

**fractal Mandelbrot** Fractal matematic denumit după cel care l-a descoperit, specialistul în știința calculatoarelor Benoit Mandelbrot.

**funcție cu apel invers** O funcție care este apelată de o altă funcție la producerea unui anumit eveniment sau în timpul unei enumerări.

**funcție de interfațare** *Vedeți clasă dispecer.*

**funcție de tratare** *Vedeți tratarea mesajelor din Windows.*

**funcție de tratare a mesajelor** *Vedeți interceptare.*

**funcție de tratare a mesajelor Windows** O funcție care este apelată de fiecare dată când aplicația primește un mesaj asociat. Funcția interpretează mesajul și poate să efectueze operații specifice funcționalității aplicației.

**funcție globală** Funcție care are domeniu global și poate fi apelată de oriunde din program.

**funcție membru protejată** Vedeți *acces protejat*.

**funcție membru publică** Vedeți *acces public*.

**funcție supradefinită** O funcție care are mai multe definiții. Vedeți și *supradefinirea operatorilor*.

**funcție virtuală** Concept specific limbajului C++, implementat prin specificarea cuvântului cheie *virtual* în definiția unei funcții. O funcție virtuală care este implementată într-o clasă de bază poate fi supradefinită prin crearea unei implementări în cadrul unei clase derivate. C++ folosește funcțiile virtuale pentru a crea polimorfismul.

**funcții de decupare** Funcții care permit definirea regiunilor ce nu vor fi afectate de funcțiile de desenare aplicate asupra lor.

**funcții dispecer** Vedeți *clasă dispecer*.

**galeria de componente (Component Gallery)** O listă a controalelor și componentelor COM înregistrate care pot fi integrate într-un proiect, accesibilă din Developer Studio.

**galeria de imagini Microsoft** O colecție de imagini care însoțește pachetul de programe Microsoft Office. Aceste imagini sunt destinate integrării în documentele și fiile de calcul create de utilizatori.

**GDI** Vedeți *interfață cu dispozitivele grafice*.

**generator de clase** O metodă de programare în C++ care permite crearea obiectelor de către alte obiecte.

**generator de numere pseudo-aleatoare** Un algoritm care generează o secvență de numere ce par a fi aleatoare; în realitate, aceste numere sunt pseudo-aleatoare, deoarece secvența poate fi reprodusă prin stabilirea aceleiași rădăcini de generare.

**GUID** Număr global unic pe 128 de biți, a cărui unicitate este garantată.

**hartă de mesaje** În MFC, un mecanism bazat pe macrodefiniții care realizează asocierea mesajelor Windows cu funcții C++.

**HMENU** Un indicator către un obiect meniu la nivelul sistemului de operare Windows.

**HRESULT** Vedeți *SCODE*.

**HTML** Hypertext Markup Language este un limbaj textual care conține instrucțiuni speciale pentru inserarea de imagini și hiperlegături, acestea din urmă permițând utilizatorului să treacă de la o pagină la alta printr-un clic de mouse. HTML este folosit preponderent în paginile World Wide Web de pe Internet.

**IID** Un identificator al unei interfete COM, un număr global unic pe 128 de biți care identifică fiecare interfață de obiect COM.

**IMPLEMENT\_DYNAMIC** Vedeți *informații dinamice de clasă*.

**implementare de clasă** Vedeți *cod de implementare*.

**înălțime de caracter** Înălțimea zonei vizibile folosită la afișarea unui caracter dintr-un font.

**încapsulare** Inserarea datelor unui document într-un alt document, de cele mai multe ori diferit.

**înclinare** Panta de afișare a caracterelor, exprimată în zecimi de grad.

**indicator** O secțiune din bara de stare a unei aplicații, care afișează text (sau o pictogramă) pentru a prezenta o anumită informație de stare sau de configurație.

**indicator** Un tip de variabilă conținând o referință unică (de regulă un număr) care identifică un obiect de un anumit tip. Spre exemplu, porumbeii sunt etichetați uneori printr-un număr prins de un picior. Echivalentul acestui număr în calculator este un indicator, iar porumbelul este obiectul referit de către indicator.

**informații de atribute** Informații care rețin culorile implicite, modurile de desenare și acțiunile care vor influența funcțiile de desenare.

**informații de tip** Informații care descriu diverșii parametri transmiși funcțiilor implementate printr-o interfață dispecer. Acestea pot fi reunite în biblioteci care se numesc biblioteci de tipuri.

**informații dinamice de clasă** Clasele MFC derivate din CObject pot conține informații care identifică clasa respectivă, prin implementarea macrodefiniției `DECLARE_DYNAMIC` și a celor asociate. În consecință, aplicațiile pot determina clasa căreia aparține un obiect folosind macrodefiniția `RUNTIME_CLASS` și funcția `IsKindOf`.

**infrastructură** Schelet de aplicație care oferă mai multe tipuri de suport pe baza cărora este posibilă dezvoltarea unei aplicații.

**infrastructură SDI** Vedeți *interfață document singular*.

**infrastructură standard** Infrastructura utilizată de aplicațiile SDI și MDI.

**instanțiere** Crearea unei instanțe a unei clase sub forma unui obiect real, aflat în memoria calculatorului.

**inter-blocare** Situația în care un proces așteaptă o resursă ocupată de un alt proces, care așteaptă la rândul său o resursă ocupată de primul proces, astfel încât nici unul dintre ele nu poate continua.

**interceptare** Scrierea unei funcții de tratare care va fi apelată ori de câte ori este recepționat un anumit mesaj sau eveniment.

**interfață cu dispozitivele grafice (GDI)** O mulțime de obiecte și metode de desenare care permit aplicațiilor să genereze reprezentări grafice ale propriilor date prin intermediul unui set generic de funcții și structuri.

**interfață de programare a aplicațiilor (API)** Biblioteci specializate de cod, concepute pentru a ajuta la efectuarea unui anumit gen de operații, de regulă în conjuncție cu tipuri specifice de hardware.

**interfață dispecer OLE** Vedeți *clasă dispecer*.

**interfață document multiplu** Arhitectură de bază standard care separă datele (numite documente) de reprezentările multiple ale acestora. Într-o aplicație MDI pot fi deschise simultan mai multe documente, în timp ce într-o aplicație SDI nu poate fi deschis la un moment dat decât un singur document.

**interfață document singular** Arhitectură de bază standard care separă datele (numite documente) de reprezentările multiple ale acestora. Într-o aplicație SDI nu poate fi deschis la un moment dat decât un singur document, în timp ce într-o aplicație MDI pot fi deschise simultan mai multe documente.

**interfață pentru controlul multimedia**

O bibliotecă API care oferă o colecție de funcții standard ușor de utilizat, în scopul redării de clipuri video sau audio pe bază de compact-discuri, fișiere de undă audio, fișiere AVI sau dispozitive de video digital.

**interferență moiré** O mulțime de linii care formează un model prin intersecția a două seturi de linii.

**intermediere** Operația de împachetare a parametrilor unei funcții într-o formă universal compatibilă pentru transmisia între procese sau calculatoare din rețea.

**iterator** O funcție care parcurge o listă de elemente, cum ar fi fonturile, apelând o funcție cu apel invers pentru fiecare element.

**izotropic** Cu proprietăți similare în direcții diferite.

**legare târzie** Informațiile de tip sunt determinate la momentul execuției; nu la compilarea programului client (legare timpurie).

**legarea și încapsularea obiectelor** Capacitatea unui program de a afișa și utiliza în cadrul propriei ferestre un alt program. Spre exemplu, o foaie de calcul poate fi afișată într-un procesor de text fără a fi necesar ca acesta să cunoască informații despre obiectul afișat; codul pentru afișarea și gestionarea obiectului aparține foi de calcul.

**.lib** Vedeți *bibliotecă cu legare statică*.

**limbaj structurat de interogare (SQL)** Un limbaj standard pentru extragerea și modificarea informațiilor din bazele de date.

**limbajul de definiție a interfețelor de la Microsoft** Un limbaj care definește interfețele și clasele COM, permițând

generarea codului pentru apelul de proceduri la distanță (RPC – Remote Procedure Call) și transmiterea parametrilor într-o manieră bine definită și compatibilă între diferitele aplicații.

**listă înlănțuită** Listă de elemente în care fiecare element conține pointeri către elementele următor și anterior.

**machetă de dialog** O resursă a unei aplicații care descrie așezarea unei mulțimi de controale în scopul de a înfățișa utilizatorului un formular interactiv.

**machetă de document** O serie de clase C++ care efectuează crearea dinamică a claselor document, reprezentare și cadru în aplicațiile SDI și MDI.

**machete** Un element suplimentar din sintaxa C++ care permite generarea automată a unor clase și funcții pe baza unor anumite tipuri de date, pornind de la secvențe de cod generic. Sunt înrudite cu macrodefinițiile.

**mapare de fonturi** Windows include o serie de funcții care selectează și creează pe baza fonturilor instalate în sistem un font care satisface cel mai bine criteriile specificate.

**maparea coordonatelor** Vedeți *mod de mapare*.

**MCI** Vedeți *interfață pentru controlul multimedia*.

**MDI** Vedeți *interfață document multiplu*.

**mediul Windows** Programele care rulează în cadrul sistemului de operare Windows și al extensiilor acestuia.

**megaocetă** Unitate de măsură pentru volumul de date. Reprezintă 1.048.576 de ocete.

**membru privat** O variabilă sau un obiect membru care este protejat astfel încât să nu poată fi accesat direct decât prin metodele aparținând aceleiași clase. Orice alte clase trebuie să efectueze accesul prin intermediul unor funcții membru de acces.

**memorie globală** Concept din Windows care denotă o zonă de memorie alocată de sistemul de operare și potențial vizibilă altor aplicații, aflată în spațiul de adrese al sistemului de operare, și nu în spațiul de adrese local al unui program.

**memorie partajată** Zonă de memorie disponibilă mai multor aplicații aflate în execuție, folosită de regulă pentru transferul informațiilor între programele care rulează. În mod normal, această memorie este izolată de alte programe pentru a împiedica alterarea accidentală a spațiului de memorie a unui alt program.

**mesaj** Vedeți *eveniment*.

**mesaj de notificare** Tip special de mesaj Windows, transmis de regulă de un control pentru a informa programul că utilizatorul a interacționat cu respectivul control într-o anumită manieră.

**mesaj de notificare reflectat** Un mesaj de notificare care a fost transmis de către fereastra părinte unui control, fiind reflectat apoi de acest control către o funcție de tratare.

**meta-fișier extins** Format de fișier care stochează o imagine într-o formă independentă de dispozitiv. Formatul extins este destinat anume aplicațiilor Win32.

**metode** Funcții care aparțin unei anumite clase.

**MFC** Vedeți *biblioteca de clase de bază Microsoft*.

**Microsoft Internet Explorer 4** Un program browser pentru documente hipertext care afișează paginile HTML și permite utilizatorului navigarea printre paginile World Wide Web.

**Microsoft QuickView** Program care însoțește Windows 95 și Windows NT, având capacitatea de a vizualiza mai multe tipuri de fișiere.

**MIDL** Vedeți *limbajul de definiție a interfețelor de la Microsoft*.

**mod de mapare** Mod al contextului dispozitiv în care se efectuează conversia între coordonatele logice și coordonatele de tip pixel, cu scopul de a genera o reprezentare la scară reală a datelor. Vedeți și *coordonate logice*.

**modelul componentelor obiectuale** Model de programare care extinde conceptul de orientare pe obiecte dincolo de domeniul unui singur program. Mai multe programe diferite pot să comunice și să folosească obiecte prin intermediul unor interfețe bine definite, fără a fi necesar să știe unde se află obiectele respective. Obiectele pot fi rulate local, într-un alt program sau pe un alt calculator aflat în rețea.

**moduri de desenare** Configurații care afectează rezultatul generat implicit de funcțiile de desenare standard.

**moștenire** Crearea unei noi clase C++ prin extinderea funcționalității unei clase existente. Vedeți și *subclasare*.

**multimedia** Combinația de dispozitive audio și video din cadrul unui calculator personal care permite înregistrarea sunetului, scanarea imaginilor și redarea compact-discurilor, a clipurilor video și a clipurilor audio.



**new** Vedeți *instanțiere*.

**nod de ramificație** Un element al unui control arbore care conține elemente subordonate.

**nod frunză** Un element dintr-un control arbore care nu are elemente subordonate.

**nod rădăcină** Elementul de la baza unui control arbore.

**nucleul Win32** Un nivel de cod (interfață pentru programarea aplicațiilor) care implementează în Windows funcționalitatea de bază a sistemului de operare.

**numerotare ordinală** Sistem pentru numerotarea distinctă a fiecărei funcții dintr-o bibliotecă DLL.

**.ocx** Tip de fișier care conține codul executabil ce implementează un control ActiveX.

**ODBC** Vedeți *conectivitate deschisă a bazelor de date*.

**șir cu terminator nul** Standard de codificare a șirurilor de caractere în care după caracterele ASCII se adaugă un caracter cu valoarea zero, indicând astfel sfârșitul șirului. În mod normal, valorile caracterelor ASCII nu coboară sub 32 (care corespunde unui spațiu), ceea ce face ca zero să fie o valoare specială.

**OLE** Vedeți *legarea și încapsularea obiectelor*.

**orientare** Înclinarea fiecărui caracter, exprimată în zecimi de grad.

**orientare pe obiect** Tehnică de programare care stabilește o legătură strânsă între comportament și proprietăți sub forma unor clase ale căror obiecte corespund obiectelor din lumea reală. Aceasta permite accelerarea dezvoltării și crearea unor aplicații mai robuste prin descompunerea

problemei în obiecte individuale, care pot fi testate separat și care comunică cu alte obiecte prin interfețe stabilite de comun acord.

**paleta sistem** Mulțime comună de culori întreținută de Windows pentru a optimiza reprezentarea culorilor standard folosite de aplicații atunci când nu este nevoie de facilități rafinate de sinteză a culorilor.

**peisaj** O imagine care are lățimea mai mare decât înălțimea.

**perioada inertă dintre cadre** Perioada care începe imediat după afișarea completă a unui cadru și durează până când raza de electroni este adusă în colțul stânga sus pentru afișarea unui nou cadru, perioadă pe durata căreia raza de electroni este stinsă.

**permanent** Date stocate pe disc astfel încât proprietățile să rămână permanente și să poată fi accesate atunci când controlul este modificat sau utilizat.

**pictogramă** O resursă de tip imagine care conține o pereche de bitmap-uri, permițând afișarea transparentă.

**pixel** Cea mai mică unitate de afișare care poate fi modificată individual.

**pointer** Un tip de variabilă care reține adresa unei alte variabile sau a unui obiect, astfel încât o funcție poate să editeze indirect un obiect transmițând adresa acestuia unei alte funcții.

**pointer de funcție** Un tip special de pointer din C++ care reține adresa de început a unei funcții.

**portret** O imagine care are înălțimea mai mare decât lățimea.

**potențial de creștere** Capacitatea de a crește în dimensiune la cerere.

**poziția cursorului grafic** O poziție de coordonate reținută în cadrul contextului dispozitiv, care reprezintă ultima poziție în care s-a desenat, astfel că o nouă operație de desenare poate fi efectuată relativ la cea anterioară.

**preprocesare** Prima etapă a procesului de compilare, care dezvoltă macrodefinițiile și urmărește directivele `#include` și `#ifdef` pentru a determina care cod va fi compilat și care nu.

**prietenos cu utilizatorul** Concept din programare care descrie o interfață de aplicație care este ușor de înțeles și de folosit.

**prin cod** Provocarea unui eveniment prin linia de cod, nu prin acțiuni ale utilizatorului.

**proiectare orientată pe obiect** Vedeți *orientare pe obiect*.

**proporția dimensiunilor** Raportul dintre lățime și înălțime. De regulă, descrie raportul dintre lățimea și înălțimea unui pixel. Dacă acest raport diferă între reprezentări ale acelorași date, relațiile de dimensiune vor diferi și ele.

**puncte de întrerupere** Marcaje care pot fi asociate liniilor de cod pentru a forța compilatorul să întrerupă execuția în dreptul acestora atunci când rularea se face prin intermediul depanatorului.

**rădăcină de generare** Vedeți *generator de numere pseudo-aleatoare*.

**referință de culoare** O valoare pe 24 de biți care conține componentele de roșu, verde și albastru ale unei anumite culori folosite la desenare.

**relație geometrică** Poziționarea unui element în raport cu altul. Mai sus, mai jos,

la stânga și la dreapta reprezintă toate relații geometrice.

**reprezentare** Reprezentare vizuală a datelor programului, afișată într-o fereastră.

**reprezentare de tip arbore** O reprezentare care afișează o ierarhie de elemente textuale, sub forma unor cataloage care conțin elemente subordonate. Aceste elemente subordonate pot fi și ele cataloage. Spre exemplu, secțiunea din stânga a programului Windows Explorer înfățișează structura de directoare a unui disc în cadrul unei reprezentări de tip arbore.

**reprezentare de tip editare formatată** O reprezentare care permite utilizatorului să editeze text și să insereze obiecte OLE. Oferă facilitățile de bază ale unui procesor de text în scopul manipulării și formătărilor textului.

**reprezentare de tip listă** O reprezentare care afișează elemente textuale, însoțite eventual de pictograme, sub forma unei liste.

**resursă** O componentă adițională a unui proiect, pe lângă codul C++, care este stocată împreună cu aplicația și poate fi folosită prin cod pentru afișarea de pictograme, meniuri, bitmap-uri, șiruri sau casete de dialog.

**resurse ale aplicației** Vedeți *resursă*.

**rezoluție** Cea mai mică unitate care poate fi reprezentată individual.

**schemă de bază de date** Structura de tabele și câmpuri a unei baze de date.

**schimb dinamic de date** Procedeu mai vechi pentru transferul datelor între aplicațiile din Windows. Deși acest mecanism poate fi folosit și acum, este de preferat să utilizați OLE.

**SCODE** O valoare pe 32 de biți folosită pentru codificarea unitară a erorilor apărute în funcțiile OLE.

**SDI** Vedeți *interfață document singular*.

**semnal de undă** Vedeți *buffer pentru semnal de undă*.

**a serializa** Vedeți *serializare*.

**serializare** Operația de transformare a unei mulțimi de variabile într-un flux continuu de date sau, respectiv, de reconstituire a mulțimii de variabile pe baza fluxului de date.

**server Gopher** Un protocol popular de server de fișiere și documente, aflat în declin după apariția World Wide Web.

**set Mandelbrot** Vedeți *fractal Mandelbrot*.

**sincronizare** Controlul asupra accesului proceselor la resurse, astfel încât un proces să acceseze resursa în timp ce celelalte așteaptă să le vină rândul.

**SQL** Vedeți *limbaj structurat de interogare*.

**stare de activare** Starea unei ferestre, care poate fi activată sau dezactivată.

**strictețea tipurilor** Specificarea în prototipuri a claselor exacte, și nu a claselor de bază, mai generice, pentru a asigura generarea unui avertisment la compilare în cazul în care o funcție primește sau întoarce un tip necorespunzător de clasă.

**subclasare** Extinderea funcționalității existente prin utilizarea unei implementări deja create și dezvoltarea acesteia.

**supradefinirea operatorilor** O clasă poate să implementeze funcții reprezentate prin operatori obișnuiți din C++, așa cum sunt =, , +=, % etc. Funcțiile sunt

implementate în mod obișnuit, dar definițiile lor includ cuvântul cheie operator. Parametrii sunt considerați a fi operandii din partea dreaptă a operatorului.

**tabelă de acceleratori** O resursă a unui proiect constând dintr-un vector de structuri de date care definesc combinațiile de taste ce au ca efect trimiterea de comenzi către aplicație.

**tabelă de șiruri** O resursă a unui proiect care conține un vector de șiruri de caractere.

**TCHAR** Definește date de tip caracter. Dacă `_UNICODE` este definit, `TCHAR` se definește ca un tip de caractere pe doi octeți (`wchar_t`); altfel, `TCHAR` se definește ca un tip de caractere pe un singur octet (`char`).

**TCP/IP (Transmission Control Protocol/Internet Protocol)** Un set de definiții standardizate care formează un protocol pentru comunicațiile de nivel scăzut în rețea și prin Internet, bazat pe transmisia de pachete de date.

**test de clic** Operație care verifică dacă utilizatorul a efectuat clic în interiorul unei anumite regiuni de pe ecran.

**Unicode** Set de caractere în care fiecare caracter este reprezentat pe câte doi octeți, făcând posibilă codificarea tuturor simbolurilor existente în lume.

**variabilă locală** Variabilă al cărei domeniu are o întindere locală, în cadrul unei funcții.

**vector** Descrie simultan direcția și mărimea (distanța).

**versiune finală** Versiune a unui program în care nu există informații pentru depanare. În mod normal, versiunea finală este cea care ajunge la utilizator.

**versiune pentru depanare** Versiune a unui proiect care conține informații pentru depanare, permițând execuția controlată și inspectarea la rulare a codului sursă.

**vtable** O tabelă de funcții virtuale, creată de compilator pentru fiecare definiție de clasă. Folosită în COM pentru a defini specificațiile unei interfețe.

**Win32** Vedeți *nucleul Win32*.

**Windows Explorer** O aplicație standard din Windows care permite utilizatorului să

vizualizeze și să manipuleze structurile de directoare și fișiere ale discurilor.

**WYSIWYG** Abreviere pentru „What you see is what you get” (“Ceea ce vezi este ceea ce obții”). Se referă la faptul că reprezentarea de pe ecran a unor date va fi identică cu reprezentarea tipărită.

**zgomot alb** Un zgomot aleator, asemănător celui produs de un radio care nu este acordat pe nici un canal.

# Index

## Simboluri

#include, directivă preprocesor, 217  
 &=, operator (stiluri ale reprezentării de tip listă), 434  
 {{AFX\_MSG\_MAP, comentariu, 76  
 \ (backslash), caracter, 550  
 \n (caracter linie nouă), 94  
 |=, operator (stiluri ale reprezentării de tip listă), 434  
 3D (Direct 3D), 673-674

## A

AbortDoc(), funcție, 525-526  
 About, casetă de dialog, 344, 607  
 abstracte, clase (tabele de funcții virtuale), 583  
 acceleratori, tabelă, 45  
 acces, funcții (reprezentare de tip listă), 428  
 accesare, 510  
   butoane de opțiune, 84-87  
   casete de dialog, 69  
   datele documentului, 261-262  
   DirectDraw, 667  
   documente, 262-264  
   MDI, aplicații, 489

spațiul de lucru al proiectului, 38  
 text  
   casete combinate, 124-125  
   casete de editare, 102-104  
 activare  
   butoane, 299-300  
   comenzi de meniu, 278  
 Active Template Library (ATL), 605  
 active, reprezentări (aplicații MDI), 481  
 ActiveX Control Wizard, 605-609  
 ActiveX, container de test al controalelor, 628-629  
 ActiveX, controale  
   About, casetă de dialog, 607  
   acțiunile utilizatorului, 612-614  
   ATL, 605  
   clase, 606-608  
   clase asociate  
     controalelor, 192  
     funcții de tratare a evenimentelor, 196-199  
     proprietăți, 194-195  
     variabilă membru a clasei dispecer, 193  
   compilare, 626-628  
   creare (clasă dispecer), 196-198

desenare, 609-612  
 documente active, 481, 602  
 editorul de resurse, 191  
 evenimente  
   adăugare, 615  
   generare, 614-617  
   tratare, 612-614  
 fișiere de asistență, 605-606  
 galeria de componente, 186-188  
 implementare, 608  
 infrastructură, 605-609  
 Insert Object, casetă de dialog, 607  
 Internet, 186  
 înregistrare, 186, 626-628  
 jurnalizarea  
   evenimentelor generate, 630  
 licențiere, 605-606, 628  
 MCI (interfață pentru controlul multimedia), 190  
 opțiuni avansate, 608  
 paleta de controale  
   adăugarea  
     controalelor, 189-190  
   testarea controalelor, 190  
 proprietăți, 617-627  
   clase asociate controalelor, 194-195

pagină pentru proprietățile generice de culoare, 619  
 permanență, 624-626  
 proprietăți ale controalelor, 191-192  
 proprietăți  
   ambientale, 628-629  
   proprietăți generice, 617-618  
   proprietăți specifice, 619, 622-624  
   proprietăți standard, 191-192  
   testare, 628  
   tipuri, 617  
   securitate, 605-606  
   subclasare, 608, 628  
   testare, 613-614, 628-630  
 actualizarea funcțiilor de tratare, 267-268  
 actualizarea reprezentărilor, 267-268  
 acumulare, tipărire, 528  
 Add a Class, casetă de dialog, 284  
 Add Event, casetă de dialog, 615  
 Add Member Function, casetă de dialog, 31, 41-43, 80, 266, 277, 494  
 Add Member Variabile, meniu (ClassView), 261-262  
 Add Member Variable, casetă de dialog, 41, 74, 80, 84-85, 108, 240, 623  
 Add Property, casetă de dialog, 617  
 AddDocTemplate(), funcție, 485  
 Adding a Class, casetă de dialog, 202  
 AddRef(), funcție, 583

AddString(), funcție, 129, 219  
 AddView(), funcție, 470  
 afișare  
   casete de editare  
     adăugare, 100-102  
     adăugarea variabilelor, 102-103  
   determinarea textului, 102-104  
   inițializarea câmpurilor, 102-104  
   mesaje de notificare, 104-105, 107  
   moștenirea claselor, 107-112  
   multi-linie, 112  
   taste mnemonice, 102  
   testare, 102  
   tratarea mesajelor, 104-105  
 ColeDateTime, 159  
 perioada inertă dintre cadre, 672  
 afișare specifică, butoane  
   clasă, 235-238  
   stări, 235-237  
 AfxGetApp(), funcție, 489  
 AfxGetMainWnd(), funcție, 302, 489  
 AfxMessageBox(), funcție, 200  
 AfxOleInit(), funcție, 560, 595-596  
 agregare, 666  
 alb, zgomot, 657  
 algoritmi, 642  
 aliniere  
   controale în casetele de dialog, 66  
   indicatori pentru barele de dialog, 306  
   text, 380-382, 384-385  
 AllocSysString(), funcție, 591-592

Ambient Properties, casetă de dialog, 629  
 ambientale, proprietăți (controale ActiveX), 617, 628-629  
 AmbientBackColor(), funcție, 611  
 AmbientForeColor(), funcție, 611  
 analiza parametrilor din linia de comandă, 255-256  
 ancorare  
   bară de controale, indicatori, 295  
   bară de dialog, 306  
   bare cu instrumente, 294  
     ferestre cadru, 295  
     indicatori, 296  
     standard, 295-297  
   ferestre, 293  
 animație  
   Direct3D, 673-674  
   DirectDraw  
     accesare, 667  
     agregare, 666  
     comutare, 668  
     crearea obiectelor, 666  
     dublu buffer, 668, 671-672  
     funcții GDI, 667  
     HAL, 665  
     HEL, 665  
     liste pentru decupare, 667  
     ModeX, 665  
     nivel de cooperare, 667  
     obiecte, 664-667  
     palete, 667  
     suprafețe, 667-668  
   DirectPlay, 674  
   modificarea dinamică a culorilor, 667

perioadă inertă, 672  
 viteză, 655-656  
**antet, fișiere**  
 definiții de clase, 534  
 elemente de meniu,  
 271-272  
 resurse meniu, 271  
**apel de proceduri la  
 distanță (RPC), 587**  
**apel invers, funcții, 393**  
**API (interfață pentru  
 programarea  
 aplicațiilor). *Vedeți și  
 interfețe; reprezentări;  
 ferestre***  
 Direct3D, 673-674  
 DirectDraw  
 accesare, 667  
 agregare, 666  
 comutarea  
 suprafețelor,  
 667-668  
 crearea obiectelor,  
 666  
 dublu buffer, 668,  
 671-672  
 funcții GDI, 667  
 HAL, 665  
 HEL, 665  
 liste pentru decupare,  
 667  
 ModeX, 665  
 modificarea dinamică  
 a culorilor, 667  
 nivel de cooperare,  
 667  
 obiecte, 664-667  
 palete, 667  
 suprafețe, 667  
 DirectInput, 674  
 DirectPlay, 674  
 DirectSetup, 674  
 DirectSound  
 blocare/deblocare,  
 658  
 buffere pentru semnal  
 de undă, 655-656  
 buffere primare, 657  
 buffere secundare,  
 657-658  
 interfață, 656-657  
 mixarea bufferelor,  
 658-659, 662-664  
 nivele de cooperare,  
 657, 663  
 redarea bufferelor,  
 658  
 repetarea bufferelor,  
 658  
 zgomot alb, 657  
**MAPI**  
 COM, 581-582  
 funcții, 674-677  
 MAPI extins, 675  
 suport în  
 infrastructură,  
 677-682  
**MCI (interfață pentru  
 controlul multimedia),  
 681, 190**  
 comenzi, 682-685  
 configurarea  
 proprietăților, 192  
 dispozitive, 682-683  
 fereastră MCI,  
 689-692  
 mesaje de notificare,  
 685-688  
 prezentare, 655  
**SDK, 655**  
 DirectX Game SDK,  
 655-656  
 Game SDK, 655  
 MAPI, 675  
 OLE DB SDK, 655  
 viteză, 655-656  
**aplicații**  
 aplicații de gestionare a  
 fișierelor (ferestre  
 multi-sectiune statice),  
 468-469

arhitectura Document/  
 Reprezentare, 246,  
 260-265, 267-268  
 asamblare/rulare, 24-25  
 baze de date  
 actualizarea datelor,  
 575  
 conectare, 570-572  
 interogări SQL,  
 572-575  
 schimb de câmpuri,  
 576  
 suport, 568-570  
 capturarea mouse-ului,  
 179-180  
 casete de dialog  
 denumire, 58-60  
 dimensionare, 62-63  
 proiectare, 54-58  
 proprietăți, 58  
 stiluri, 59-60  
 ClassView, 80  
 clienți de automatizare  
 clasă driver de  
 dispecer, 599-601  
 creare, 599-601  
 înregistrarea  
 obiectelor, 600  
 compilare/legare, 24-25,  
 368  
 configurații, 24-25, 632  
 controale din casete de  
 dialog  
 aliniere, 66  
 dimensionare, 64  
 dispunere, 60, 62-64  
 editare, 65  
 grupare, 68-69  
 linii de ghidare,  
 66-67  
 ordinea de selectare,  
 69-70  
 taste de acces, 70-71  
 Fire, 141-142

IDD\_MOUSEMSG\_DIA  
 LOG, 169  
 interfețe  
 conectarea codului,  
 30, 32  
 controale buton, 26,  
 28-29  
 Mail and Fax, panoul de  
 control, 675-676  
 manipularea fișierelor,  
 aplicații de tip dialog,  
 532  
 MDI (interfață document  
 multiplu), aplicații  
 bare cu instrumente,  
 482  
 bare de stare, 482  
 clase, 480-482  
 creare, 477-480  
 date, 490-496  
 documente, 479, 482,  
 485-489  
 inițializare, 483-486  
 machete de document,  
 483-486, 495-502  
 meniuri, 482  
 reprezentări,  
 481-483, 493-496  
 resurse, 481-483  
 serializare, 532  
 tabele de șiruri, 482  
 tipărire, 504  
 variabile membru,  
 490-492  
**ODBC**  
 distribuire, 564-565  
 suport, 569  
 pictograma MFC  
 implicită, 225  
 reprezentare de tip  
 înregistrare  
 controale de editare,  
 578-579  
 creare, 576-577  
 editare, 577-578

rulare, 26  
 SDI (interfață document  
 singular)  
 adăugarea opțiunilor  
 de meniu, 265-266  
 bare cu instrumente,  
 253  
 bară de stare, 253  
 clase, 250-251  
 creare, 246-249  
 elemente ale ferestrei,  
 251-253  
 funcții, 255-260  
 funcții ale  
 infrastructurii  
 Document/Reprezent  
 are, 255-260  
 machete de document,  
 253-260, 265-266  
 manipularea  
 fișierelor, 532-534  
 pagina  
 ResourceView, 253  
 reprezentare de tip  
 arbore, 441  
 reprezentare de tip  
 listă, 427  
 reprezentări de tip  
 formular, 404  
 serializare, 532-534  
 tipărire, 504  
 servere de automatizare  
 adăugarea de metode,  
 597-599  
 biblioteci de tipuri,  
 598  
 COleObjectFactory,  
 595-596  
 containere, 602-604  
 creare, 591-597  
 DISP\_FUNCTION,  
 macrodefiniție, 597  
 documente active, 602  
 încapsulare, 602-604  
 limbaj de definiție a  
 obiectelor, 593-596  
 nume, 591-592  
 parametri în linia de  
 comandă, 595-596  
 registru, 592  
 rularea în fundal, 592  
 tabela obiectelor  
 aflate în rulare, 592  
 VB Scripting,  
 591-592  
 Word, 589  
 WordPad, 603  
 testare, 32  
**AppendMenu(), funcție,  
 286-287**  
**ApplyColor(), funcție, 308**  
**AppWizard**  
 arbore, tip de  
 reprezentare, 441  
 bara cu instrumente  
 standard a  
 infrastructurii, 292  
 bare suport, 319-321  
 butoane de comandă, 72  
 activare/dezactivare,  
 78-79  
 afișare/ascundere,  
 76-78  
 afișarea în titlul  
 dialogului, 80-82  
 funcții de tratare  
 pentru clic, 79-82  
 hărți de mesaje,  
 75-76  
 la momentul  
 execuției, 76-79, 82  
 proprietăți, 73-74  
 tratarea  
 evenimentelor, 74  
 butoane de opțiune, 82  
 accesare, 84-87  
 grupare, 83-85  
 casete de dialog  
 denumire, 58-60  
 dimensionare, 62-63  
 proiectare, 54-58

proprietăți, 58  
 stiluri, 59-60  
 casete de validare  
   stare, 89-92  
   tri-stare, 88  
 comentarii, 56  
 formular, tip de  
   reprezentare în aplicații  
   SDI, 404  
 listă, tip de reprezentare  
   în aplicații, 427  
 opțiuni de bază, 23  
 ReadMe.txt, fișier, 325  
 secțiunea spațiului de  
   lucru, 24-25  
 tipărire/previzualizare,  
   504  
**apăsat, stare, butoane**  
**bitmap, 235-237**  
**arbore, control, 238-240,**  
**243, 469**  
   creare, 116-117  
   elemente, asociere, 442  
   noduri, 116  
   populare, 125-128  
   rânduri, selectare, 442  
   tipuri de liste cu imagini,  
     242  
   variabile, 117  
**arbore, tip de reprezentare**  
 AppWizard, 441  
 arbore, control, 469  
 editare imediată, 447-449  
 inserarea elementelor,  
   442, 445  
 nod selectat, determinare,  
   446-448  
 stiluri, 442  
**arhitecturi,**  
**Document/Reprezentare,**  
**246, 260-265, 267-268**  
**asistenți**  
 ActiveX Control Wizard,  
   605-609

AppWizard  
   aplicații cu  
     reprezentare de tip  
     listă, 427  
   aplicații SDI cu  
     reprezentare de tip  
     formular, 404  
   arbore, tip de  
     reprezentare, 441  
   bara cu instrumente  
     standard a  
     infrastructurii, 292  
   bare suport, 319-321  
   butoane de comandă,  
     72-82  
   butoane de opțiune,  
     82-88  
   casete de dialog,  
     54-64  
   casete de validare,  
     88-92  
   comentarii, 56  
   opțiuni de bază, 23  
   ReadMe.txt, fișier,  
     325  
   secțiunea spațiului de  
     lucru, 24-25  
   tipărire/previzualizar  
     e, 504  
 ClassWizard, 262  
   ActiveX, tratarea  
     evenimentelor,  
     196-199  
   Adding a Class,  
     casetă de dialog,  
     284  
   bare de dialog,  
     controale, 307  
   BN\_CLICKED,  
     tratare, 325  
   CDialog, clase,  
     201-203  
   comentarii, 76  
   funcții de tratare  
     pentru comenzile de  
     meniu, 276-278, 284

funcții de tratare  
   pentru interfața cu  
   utilizatorul, 278  
 maparea variabilelor  
   peste controale de  
   editare, 410  
 prezentare, 84-85  
 regenerare, 50  
 variabile membru,  
   193

## asistență

comentarii AppWizard,  
   56  
 controale ActiveX,  
   605-606

asocierea câmpurilor, 576  
**ASP (Active Server Pages),**  
**186**

**ASSERT, macrodefiniție,**  
**642-644**

**ASSERT\_KINDOF(),**  
**macrodefiniție, 474-475**  
**ASSERT\_NULL\_OR\_POI**  
**NTER, macrodefiniție,**  
**643**

**ASSERT\_POINTER,**  
**macrodefiniție, 643**

**ASSERT\_VALID(),**  
**macrodefiniție, 431**

**ATL (biblioteca de**  
**machete active), 605**

## atribute

CFileStatus, 554  
 fișiere, 553-556

**Attach(), funcție, 286**

**AttachClipboard(), funcție,**  
**562**

**AttachDispatch(), funcție,**  
**601**

## audio

DirectSound  
   blocare/deblocare,  
     658  
   buffere pentru semnal  
     de undă, 655-656

buffere primare, 657  
 buffere secundare,  
   657-658

interfață, 656-657  
 mixarea bufferelor,  
   658-659, 662-664

nivele de cooperare,  
   657, 663

redarea bufferelor,  
   658

repetarea bufferelor,  
   658

zgomot alb, 657

MCI (interfață pentru  
 controlul multimedia),  
 681

comenzi, 682-685  
 dispozitive, 682-683

fereastră MCI,  
   689-692

mesaje de notificare,  
   685-688

## automatizare (automatizare OLE)

clienți de automatizare

  clasă driver de  
   dispecer, 599-601

  creare, 599-601

  înregistrarea  
   obiectelor, 600

interfață dispecer,  
   589-590

servere de automatizare  
   adăugarea de metode,  
     597-599

  biblioteci de tipuri,  
     598

  COleObjectFactory,  
     595-596

  containere, 602-604

  creare, 591-597

  DISP\_FUNCTION,  
     macrodefiniție, 597

  documente active, 602

  încapsulare, 602-604

limbaj de definiție a  
 obiectelor, 593-596  
 nume, 591-592

parametri în linia de  
 comandă, 595-596

registru, 592

rularea în fundal, 592

tabela obiectelor  
   aflate în rulare, 592

VB Scripting,  
   591-592

Word, 589

WordPad, 603

suport, 590-529

## B

**backslash (\), caracter, 550**  
**bara de titlu, indicarea**  
**butoanelor de comandă,**  
**80-82**

**bară de agățare, 294**

**bară de derulare, control**

  adăugare în casete de  
   dialog, 142-143

  inițializare, 144-146

  intervale, 144

  mesaje de notificare, 149

  funcții de tratare,  
     adăugare, 146

  indicatori de tratare,  
     151-152

  listing, 149

  poziții

    determinare, 420

    negative, 151

  proprietăți, 143

  săgeți, dezactivare, 145

  variabile, mapare,  
     143-144

**bară de secționare,**  
**opțiune, casetă de dialog,**  
**460**

**bare cu instrumente**

  ancorare

  ferestre cadru, 295

  margini, 294  
   standard, 295-297  
   valori de indicatori,  
     296

ascundere/afișare, 302

bare suport, 319

butoane

  activare/dezactivare,  
     299-300

  apăsătridicat, stare,  
     299-300

  bitmap-uri, 294

  combinații de taste,  
     299

  deplasare, 299

  editor de resurse,  
     297-299

  funcții de tratare,  
     299-300

  plate, 294

  ridicate, 294

  separatori, 299

  ștergere, 299

combinații de taste, 303

Controls, bară cu  
 instrumente (Developer  
 Studio), 60, 62-64

creare, 299-302

  aspect plan, 301

  proiectare, 223

deplasare, 294

Dialog, bară cu  
 instrumente (Developer  
 Studio), 60, 62-64

dimensionare, 294

Explorer, bară cu  
 instrumente, 294

extensii, clase, 297

ferestre cadru, 295,  
   301-302

flotante

  comentarii în cod,  
     295

  dimensionare, 294

  inserare, 301

Internet Explorer, bare suport, 319-321  
 localizare, 294  
 MDI, aplicații, 482  
 poziție, 303  
 resurse, 47, 300  
 salvarea stării, 303  
 SDI, aplicații, 253  
 standard  
   *ancorare*, 295-297  
   *bară de controale*, indicatori de stil, 294-295  
   *creare*, 293, 295  
   *fereastră asociată*, 294  
   *infrastructură*, 292-299  
   *OnCreate()*, funcție, 293

**bare de controale**  
 focus, 300  
 indicatori, 294-296

**bare de secționare**  
 adăugare, 460  
 personalizare, 464-465

**bare de stare**  
 etichete explicative, 310  
 indicatori, 310, 317  
   *poziția mouse-ului*, 312  
   *proprii*, 312, 313-314  
 MDI, aplicații, 482  
 personalizare, 310-320  
 poziția mouse-ului, 318  
 SDI, aplicații, 253  
 secțiuni  
   *dimensiune*, 315-321  
   *funcții*, 315  
   *indicator*, 317  
   *indicatori de stil*, 316  
 separatori, 312, 313-314  
 stil, 315-321  
 text, 315-321

**bare suport (Internet Explorer)**  
 ancorare, 319  
 AppWizard, 319-321  
 bitmap de fundal, 319-321  
 controale încapsulate, 320  
 stiluri, 319-321  
 titlu, 321

**baze de date**  
 aplicații  
   *actualizarea valorilor*, 575  
   *conectare*, 570-572  
   *interogări SQL*, 572-575  
   *mulțimi de înregistrări*, 569-570  
   *schimb de câmpuri*, 576  
   *suport (AppWizard)*, 568-570

baze de date obiectuale, 564

baze de date relaționale  
   *configurarea surselor de date*, 566-568  
   *definire*, 564  
   *ODBC*, 564-565  
   *RDBMS*, 564-565  
   *schemă*, 564-565  
   *SQL*, 565-566  
   *User DSN/System DSN*, 568  
 depanare, 575  
 DMS (sisteme de gestionare a bazelor de date), 564  
 scheme, 534

**BCHAR, tip de date**, 396

**BeginPaint(), funcție**, 334

**BEGIN\_MESSAGE\_MAP, macrodefiniție**, 76

**Bezier, curbe**, 362-369

biblioteci (biblioteci de tipuri), 594, 598

**bItalic, parametru**, 389

**BitBlt(), funcție**, 335-336, 667-668

**Bitmap Properties, casetă de dialog**, 228

**bitmap-uri**, 46  
 butoane, 235, 304  
   *bară cu instrumente*, 294  
   *clasă buton*, 235-238  
   *stări*, 235-237

comparate cu pictograme, 225

compilate, 368

controale imagine, 231

creare, 223

dimensiuni, contexte  
   dispozitiv de memorie, 335

editarea dimensiunilor și a culorilor, 228-229

importare, 229-230

inserare, 227-228

încărcare la momentul execuției, 231-236

liste de imagini, 240-244

pensule, 367-369, 372-374

resurse, 227-228, 368

**Blob, obiect de date**  
 definiția clasei, listing, 535-536  
 implementarea clasei, 537-538

blocarea bufferelor, 658

**Blt(), funcție**, 667, 672

**.bmp, extensie de fișier**, 50

**BN\_CLICKED, tratare**, 80, 325

**Breakpoints, casetă de dialog**, 644

**browser HTML, tip de reprezentare**, 454-455

**bStrikeOut, parametru**, 389

**buffere**  
 blocare/deblocare, 658  
 buffere primare, 657  
 buffere secundare  
   *creare*, 657-658  
   *mixare*, 658-659, 662-664  
   *redare*, 658  
   *repetare*, 658  
 depășire la citire, 548-549  
 dublu buffer, 667-668, 671-672  
 mixare, 663  
 șiruri, 124  
 zgomot alb, 657

**Build, meniu, comanda Set Active Configuration**, 632

**bUnderline, parametru**, 389

**butoane**  
 adăugarea de cod, 30  
 bare cu instrumente  
   *activare*, 299-300  
   *apăsă/ridicat, stare*, 299-300  
   *aspect plan*, 294  
   *bitmap-uri*, 294  
   *combinații de taste*, 299  
   *dezactivare*, 299-300  
   *editorul de resurse*, 297-299  
   *funcții de tratare*, 299-300  
   *poziționare*, 299  
   *prescurtări de taste*, 303  
   *ridicat*, 294  
   *separatori*, 299  
   *ștergere*, 299

bitmap, 304  
   *clasă buton*, 235-238  
   *stările butoanelor*, 235-237

butoane de opțiune, 82  
   *accesare*, 84-87  
   *grupare*, 83-85

butoanele mouse-ului, 169-174  
   *dublu clic, eveniment*, 174  
   *interceptare*, 169  
   *mesaje, funcții de tratare*, 169-173

ca ferestre, 76

casete de validare  
   *stare*, 89-92  
   *tri-stare*, 88

CDC, clasă, 325

controale, 26-27, 28-29

prezentare, 72

**Buttons, proiect de tip casetă de dialog**, 72-74

**butoane de comandă**, 72  
 activare/dezactivare, 78-79  
 afișare/ascundere, 76-78  
 afișarea pe titlul dialogului, listing, 80-82  
 funcții de tratare a clicurilor, 79-82  
 hărți de mesaje, 75-76  
 momentul execuției, 76-79, 82  
 proprietăți, 73-74  
 tratarea evenimentelor, 74

**butoane de opțiune**, 82  
 determinare, 84-87  
 grupare, 83-85

**butoane grafice**  
 clasă buton, 235-238  
 stările butoanelor, 235-237

## C

**cadre**  
 adăugare de reprezentări, 470  
 ancorare, 293  
 bare cu instrumente  
   *ancorare*, 294-296  
   *inserare*, 301-302  
 bare de dialog, 305-306  
 dinamice, ferestre, 470-475  
 dinamice, ferestre multi-secțiune  
   *creare*, 458-459  
   *inițializare*, 461-463  
 imagine, controale, 231  
 multi-secțiune, ferestre, 458  
 statice, ferestre multi-secțiune  
   *creare*, 464  
   *crearea aplicațiilor de gestionare a fișierelor*, 468-469  
   *inițializare*, 464-467

**cadru elastic**, 65, 182-184

**cale de directoare**, 554

**calendar, controale**  
 adăugare în casetele de dialog, 162-164  
 inițializare, 164-165  
 interval, 165-166  
 mesaje de notificare, 166-167  
 stiluri, 163  
 variabile, 164

**Calendar, control**, 188

**Call Stack, fereastră (depanator)**, 647-648

**canale rectangulare (controale glisor)**, 152

**caractere**, 387

**CArchive, clasă**, 533

**caroiaj. Vedeți** hașură

**CArray, clasă, 260****casete combinate**

controale, 238-240, 243  
 creare, 115-118  
 derulare, 114-115  
 determinarea  
 conținutului, 124-125  
 imagini, 116  
 liste derulante, 114-115  
 mesaje de notificare,  
 124-125  
 populare, 121-124  
 sortare, 115-116  
 variabile, 115-116

**casete de editare**

adăugarea variabilelor,  
 100-103  
 controale

*câmpuri din mulțimi  
 de înregistrări,  
 578-579*

*opțiuni pentru  
 derulare, 409-410*  
*taste mnemonice, 95*

distincția  
 minuscule/majuscule,  
 100

extragerea textului,  
 102-104

inițializarea câmpurilor,  
 102-104

mesaje de notificare,  
 104-105, 107

moștenirea claselor,  
 107-112

multi-linie, 112

numerice, 100

parole, 100

taste mnemonice, 102

testare, 102

tratarea mesajelor,  
 104-105

**casete de validare**

controale, 69  
 stare, 89-92

tri-stare, 88

**casetă cu listă, control**

adăugare în caseta de  
 dialog About, 344  
 arbore, subclase, 117  
 creare, 118-119  
 mesaje de notificare,  
 130-131  
 funcții membru, 106  
 populare, 128-130  
 variabile, 119

**căutări**

fișiere, 127

interogări SQL

*actualizarea  
 valorilor, 575*

*aplicații de baze de  
 date, 572-575*

*schimb de câmpuri,  
 576*

**câmpuri**

asociere, 576

inițializare, 102-104

**CBitmap, clasă, 369****CBitmapButton, clasă,  
235-238****CBrush, clasă, 365****CButton, clasă, 89****CButtonsDlg, clasă, 74****CChildFrame, clasă, 481****CClientDC, clasă, 329****CCmdTarget, clasă, 75-76****CCommandLineInfo,  
clasă, 255, 486****CDaoDatabase, clasă, 571****CDaoRecordset, clasă, 571****CDatabase, clasă, 570-572****CDataExchange, obiect,  
208, 212****CDC, clasă, 324-329,  
529-530****CDialog, clasă**

derivare prin

ClassWizard, 201-203

OnOK(), funcție, 56-57

**CDialogBar, obiect,  
304-305****CDocument, clasă, 246,  
257-258, 260-261****CEdit, clasă**

CInitials, subclasă,  
 107-112

funcții membru, 106

**CEditView, clasă, 249,  
464-465, 470****celule, 387****CenterPoint(), funcție, 358,  
420****cercuri**

penițe, 360-362

pensule, 374-375

**CFile, clasă, 546**

clase derivate, 555-556

clasă de încapsulare,  
 545-546

indicatori pentru  
 deschidere, 548

**CFileException, obiect, 547****CFileStatus, clasă, 553-554****CFontDialog, clasă,  
397-398****CFormView, clasă, 249,  
576-577****CFrameWnd(), funcție,  
489****CFrameWnd, clasă, 251****ChangeDialogTitle(),  
funcție, 79****CheckMenuItem(), funcție,  
289****Choose Font, casetă de  
dialog, 397-398****Chord(), funcție, 375****CHtmlView, clasă, 249,  
454****CImageList, obiect,  
238-242****CInitials, clasă, 107-112****citirea fișierelor, 548-551****CityBreak, proiect de tip  
dialog, 83-88****clase**

asertiuni, 474-475

CArchive, 533

CArray, 260

CBitmap, 369

CBitmapButton, 235-238

CBrush, 365

CButton, 89

CButtonsDlg, 74

CChildFrame, 480-481

CClientDC, 329

CCmdTarget, 75-76

CCommandLineInfo,  
 255, 486

CDaoDatabase, 571

CDaoRecordset, 571

CDatabase, 570-572

CDC, 324-329, 529-530

CDialog, 201-203

CDocument, 246,  
 254-255, 260-261

CEdit, 106

CEditView, 249,  
 464-465, 470

CFile, 546

CFile, derivate, 555-556

CFont, 386

CFontDialog, 397-398

CFormView, 249,  
 576-577

CFrameWnd, 251

CHtmlView, 249, 454

CInitials, 107-112

clase abstracte, 583

clase de bază, 41

clase derivate, 41

clase dialog, 200-203

*afișarea casetelor de  
 dialog modale,  
 204-205*

*inițializare, 203-204*

*resursă machetă de  
 dialog nouă, 201*

*variabile membru,  
 206-207*

**clase dispecer**

*crearea controalelor  
 ActiveX, 196-198*

*nume de drivere, 599*

**clase document**

*funcții membru  
 publice, 481*

*stocarea datelor,  
 490-492*

*variabile membru  
 protejate, 261-262*

**CLeftView, 468****CListCtrl, 121****CListVDoc, 427-428****CListView, 249, 468****CLSID (identificator de  
clasă), 584, 591-592****CMainFrame, 251, 481****CMDIChildWnd, 481****CMDICoinApp, 480****CMDICoinDoc, 481****CMDICoinView, 481****CMDIFrameWnd, 481****CMouseMsgDlg, 170****CMultiDocTemplate,  
484****CObList, 260****colecție, 427-428****COleDateTime**

*extragerea valorilor,  
 160*

*formatarea datelor,  
 158*

*manipularea  
 obiectelor, 167*

*validitate, 157*

**COleSafeArray, 591****controale ActiveX, 192**

*About, casetă de  
 dialog, 607*

*denumire, 606-608*

*funcții de tratare a  
 evenimentelor,  
 196-199*

*Insert Object, casetă  
 de dialog, 607*

*proprietăți, 194-195*

*variabilă membru a  
 clasei dispecer, 193*

**CPaintDC, 330****CPen, 349-350****CPoint, 358****cpp, fișiere, 536****CPreviewDC, 518****CPrintDialog, 521****CPropertyPage, 204****CPropertySheet, 204**

*crearea dinamică,  
 254-255*

**CRecentFileList,  
545-546****CRecordset, 571**

*actualizarea  
 valorilor, 575*

*depanare, 575*

*interogări, 572-575*

*schimb de câmpuri,  
 576*

**CRecordView, 249,  
576-577****CRect, 178, 359****CRectTracker, 181**

*cadru elastic,  
 182-184*

*indicatori de stil, 182*

*personalizare, 183*

**CRichEditView, 249,  
450****CRotaryCtrl, 606****CScrollView, 249****CSDICoinApp, 250****CSDICoinDoc, 250****CSDICoinView, 251****CShapesDlg, 240-241****CSimpTextView, 380****CSingleDocTemplate,  
clasă, 253****CSplitterWnd, 458-459****CStatic, 231-232**



CString, 591-592  
 CStringList, 130, 427-428  
 CTreeCtrl, 118, 125  
 CTreeView, 249, 468-469  
 CUsingDBSet, 572-573  
 CView, 249, 508-509  
 CWinApp, 480  
 CWnd, 77  
   *RedrawWindow()*, funcție, 330  
   *SetWindowText()*, funcție, 78  
 DAO (ODBC), 571  
 definiții  
   afișare, 41  
   blob, definiția obiectului, 535-536  
   fișiere de antet, 534  
 funcții  
   adăugare, 41-43  
   funcții de tratare moștenite, 110  
   tip, 43  
   virtuale, 543  
 IAutoserver, 599  
 MapMode, 339  
 MDI, aplicații, 480-482  
 membri  
   inspectare, 41  
   validare, 43  
 moștenire, 107-112  
 referințe, 41  
 reprezentare, 464-465  
 SDI, aplicații, 250-251  
 serializabile  
   declarare, 534-536  
   implementare, 536-539  
 subclasare. *Vedeți* subclasare  
 variabile  
   adăugare, 41  
   denumire, 42

  incrementare, 42-43  
   inițializare, 42  
 clase de bază, 41  
 clase de colecție, 427-428  
 clase, identificatori, 584, 591-592  
 ClassView (fereastra spațiului de lucru al proiectului), 38-44, 380  
   Add Member Variable, meniu, 261-262  
   dublu clic, efectuare asupra elementelor, 39  
   inserarea de variabile și funcții membru, 80  
   meniuri de context  
     afișare, 40-41  
     inspectarea fișierelor, 41  
     opțiuni, 40-43  
   pictograme, 39  
 ClassWizard, 262  
 ClassWizard, casetă de dialog, 84-85, 233, 464-465  
 ClassWizard, comandă (meniul View), 84-85, 464-465  
   ActiveX, tratarea evenimentelor, 196-199  
   asamblare, 50  
   bare de dialog, controale, 307  
   BN\_CLICKED, tratare, 325  
   CDialog, clase, 201-203  
   Class, casete de dialog, 284  
   comentarii, 76  
   comenzi de meniu, funcții de tratare, 276-278, 284  
   interfața cu utilizatorul, funcții de tratare, 278  
   maparea variabilelor peste controale de editare, 410

  prezentare, 84-85  
   variabile membru, 193  
 ClearTics(), funcție, 153  
 CLeftView, clasă, 468  
 clienți  
   clienți de automatizare, 599-601  
   contexte dispozitiv, 329  
   intermediere COM (biblioteci de intermediere), 588  
 Clipboard  
   copiere, 558-560  
   formate, 556-558  
   lipire, 560-562  
   OLE, 556  
   OnEditCopy(), funcție, 558  
   RegisterClipboardFormat(), funcție, 556-558  
   serializare, 558-559  
 CListBox, clasă, 117  
 CListCtrl, clasă, 121, 132  
 CListVDoc, clasă, 427-428  
 CListView, clasă, 249, 468  
 Close(), funcție, 550  
 CLSID (identificatori de clasă), 584, 591-592  
 CLSIDFromString(), funcție, 592  
 clw, extensie de fișier, 50  
 CMainFrame, clasă, 251, 481  
 CMainFrame, definiția clasei, 293  
 Cmci, variabile membru de tip, 194  
 CMDIChildWnd, clasă, 481  
 CMDICoinApp, clasă, 480  
 CMDICoinDoc, clasă, 481  
 CMDICoinView, clasă, 481  
 CMDIFrameWnd, clasă, 481  
 CMenu, obiect, 285-286  
 CModeless(), funcție constructor, 214

CModeless, clasă, 218  
 CMouseMsgDlg(), funcție, 181  
 CMouseMsgDlg, clasă, 170  
 CMultiDocTemplate, clasă, 484  
 COBArray, 541  
 COBList, clasă, 260  
 CoCreateInstance(), funcție, 587, 592, 666  
 CoCreateInstanceEx(), funcție, 587  
 cod  
   AppWizard, comentarii, 56  
   conectarea la interfață, 30, 32  
   generat de ClassWizard, 203  
   limbaj de asamblare, cod (fereastra Disassembly), 649  
   separarea resurselor, 45  
 cod în limbaj de asamblare, 649  
 coduri de taste (taste virtuale), 112  
 CoGetClassObject(), funcție, 587  
 CoInitialize(), funcție, 671  
 COleDateTime, clasă  
   extragerea valorilor, 160  
   formatarea datelor, 158  
   manipularea obiectelor, 167, 198  
   validitate, 157  
 COleObjectFactory, 595-596  
 COleSafeArray, clasă, 591  
 COleVariant, obiect, 455  
 Collate, casetă de validare (casetă de dialog Print), 519  
 coloane  
   controale listă, 132-136

  reprezentare de tip listă, 434, 437-438  
 ColorChange(), funcție, 309  
 Colors, paletă, 223-224  
 COM (modelul componentelor obiectuale)  
   ActiveX, controale, 186  
   agregare, 666  
   clienți, intermediere (biblioteci de intermediere), 588  
   COM+, 581  
   convenții de denumire, 599  
   crearea obiectelor, 586-588  
   DCOM, 587  
   depanarea obiectelor, 639  
   interfețe, 581  
   interfețe duale, 590  
     CLSID, 584  
     convenții de denumire, 582  
     GUID, 584-586  
     IClassFactory, 587  
     IID, 584  
     IUnknown, 583  
     versiuni, 588-589  
 MAPI, 581-582  
 obiect de clasă, 586-588  
 obiecte, eliberarea memoriei, 584  
 prezentare, 581-582  
 server, procese  
   contexte, 586-588  
   server aflat la distanță, 586  
   server in-proces, 586  
   server local, 586  
 tabele de funcții virtuale, 582  
 combinații de taste  
   bare cu instrumente, 303

  casete de dialog, 70-71  
   casete de editare, 95, 102  
   depanarea proiectelor, 640  
   dezactivare, 95  
   listă, 95  
   mnemonice, 95, 102  
   specificare pentru elementele de meniu, 274-275  
 combinații de taste (butoane de pe bara cu instrumente), 299  
 Combo Box Properties, casetă de dialog, 114-115  
 comentarii  
   bare cu instrumente flotante, 295  
   ClassWizard, 76  
 comenzi  
   Build, meniu, Set Active Configuration, 632  
   Edit, meniu, Set Ambient Properties, 628  
   EnableDocking(), funcție, 295  
   identificatori, atribuirea resurselor meniu, 273  
   Image, meniu, Delete, 227  
   Insert, meniu, Resource, 226, 229  
   interfața cu utilizatorul, funcții de tratare, 265-269  
     activare/dezactivare, 278  
     adăugare, 276-279  
     creare, 265-266  
     editare, 267-268  
     elemente de meniu de tip comutator, 279  
     etichetele elementelor de meniu, 280  
   Layout, meniu  
     Tab Order, 84-85

Test, 84-85, 102, 614  
 MCI, 682-685  
 Tools, meniu, MFC  
 Tracer, 575  
 View, meniu  
   ClassWizard, 84-85, 233, 464-465  
   Properties, 226-228, 232  
   Resource Symbols, 56-57  
   Split, 458-459  
 Window, meniu, Split, 458-459

**COMMAND, mesaj, 276, 397-398**

**compatibilitatea înapoi (interfețe COM), 588-589**

**compilare, 24-25**  
 controale ActiveX, 626-628  
 depanarea proiectelor, 635  
 moduri de configurație, 632-633

**Components and Controls Gallery, casetă de dialog, 460**

**compuse, fișiere, 589, 595-596**  
 moniker, 603  
 serializare, 602

**comutatoare, elemente de meniu, 279-280**

**conectivitate**  
 ODBC, 564-565  
   configurarea surselor de date, 566-568  
   User DSN/System DSN, 568  
 SQL, 565-566

**configurație de asamblare, 24-25, 50**

**configurații**  
 casete de dialog, 59-60

configurație pentru versiune finală, 632

configurații de compilare, 632-633

creare, 52

proiecte, 632

surse de date, 566-568

**Confirm Classes, casetă de dialog, 188**

**containere (OLE), 602-604**

**context, meniuri**  
 adăugare, 280-282  
 afișare, 281-283  
 ClassView  
   Add Member  
     Function, opțiune, 43  
   Add Member  
     Variable, opțiune, 41  
   afișare, 40-41  
   Base Classes, opțiune, 41  
   Called By, opțiune, 41  
   Calls, opțiune, 41  
   Derived Classes, opțiune, 41  
   Go to Definition, opțiune, 40  
   Group By Access, opțiune, 41  
   inspectarea fișierelor, 41  
   References, opțiune, 40

funcție de tratare, 281-282

tratarea comenzilor din meniurile contextuale, 284

**contexte (procese server), 586-588**

**contexte dispozitiv, 323**  
 CDC, clasă, 324-329  
 coduri escape, 529-530

contexte de memorie, 335-336

contexte de redesenare, 330, 332-335

desenare, 355

fereastră client, indicator, 329

imprimantă, 526

independență de driver, 504

mapare, moduri, 337-340  
   caracteristicile dispozitivului, 342-347  
   scalare liberă, 340-342

MCI, 683

modificare, 525

penițe, 352-353

pensule, 371-372

SetBkColor(), funcție, 382

SetMapMode(), funcție, 337-339

SetTextColor(), funcție, 382

spațiu de coordonate, 337-339

TextOut(), funcție, 379

tipuri, 323

tipărire, 510-512

**controale**  
 activare/dezactivare dinamică, 151  
 ActiveX  
   About, casetă de dialog, 607  
   acțiunile utilizatorului, 612-614  
   clase, 606-608  
   compilare, 626-628  
   creare în clase  
   dispecer, 196-198  
   desenare, 609-612

fișiere de asistență, 605-606

fișiere sursă, 627

generarea evenimentelor, 614-617, 629

implementare, 608

Insert Object, casetă de dialog, 607

interfața pentru proprietăți, 617-627

Internet, 186

înregistrare, 626-628

licențiere, 605-606, 628

maparea proprietăților, 622-624

opțiuni avansate, 608

pagina pentru proprietăți generice de culoare, 619

permanenta proprietăților, 624-626

proprietăți generice, 617-618

proprietăți specifice, 619, 622-624

securitate, 605-606

subclasare, 607

testare, 613-614, 628-630

tratarea evenimentelor, 612-614

bare de dialog, 304, 307-308

bare suport, 319-321

bară de derulare  
   adăugare în casetele de dialog, 142-143  
   inițializare, 144-146  
   intervale, 144  
   mesaje de notificare, 146-150

poziții negative, 151

proprietăți, 143

săgeți, dezactivare, 145

variabile, 143-144

butoane, 72

butoane de comandă, 72  
   activare/dezactivare, 78-79  
   afișare/ascundere, 76-78  
   afișarea în titlul dialogului, 80-82  
   funcții de tratare a clicurilor, 79-82  
   hărți de mesaje, 75-76  
   momentul execuției, 76-79, 82  
   proprietăți, 73-74  
   tratarea evenimentelor, 74

butoane de opțiune, 82

determinare, 84-87

grupare, 83-85

cadru elastic, 65

**casete combinate**  
 creare, 115-118  
 derulante, 114-115  
 determinarea conținutului, 124-125  
 imagini, 116  
 mesaje de notificare, 124-125  
 populare, 121-124  
 sortare, 115-116  
 variabile, 115-116

**casete de dialog**  
 aliniere, 66  
 aspect, 60, 62-64  
 casete de validare, 69  
 dimensionare, 64  
 editare, 65  
 grupare, 68-69

linii de ghidare, 66-67

localizare, 154

navigare, 69

ordinea de selectare, 69-70

taste de acces, 70-71

**casete de editare**  
 adăugare, 100-102  
 determinarea textului, 102-104  
 distincția minuscule/majuscule, 100  
 inițializarea câmpurilor, 102-104  
 mesaje de notificare, 104-105, 107  
 moștenirea claselor, 107-112  
 multi-linie, 112  
 numerice, 100  
 opțiuni de derulare, 409-410  
 parole, 100  
 taste mnemonice, 95, 102  
 testare, 102  
 tratarea mesajelor, 104-105  
 variabile, 102-103

**casete de grupare, 68**

**casete de validare**  
 stare, 89-92  
 tri-stare, 88

câmpuri din mulțimi de înregistrări, 578-579

**controale arbore, 469**  
 adăugare de variabile, 117  
 creare, 116-117  
 linii, selectare, 442  
 noduri, 116  
 populare, 125-128  
 controale calendar

- adăugare în casetele de dialog, 162-164
- inițializare, 164-165
- interval, 165-166
- mesaje de notificare, 166-167
- stiluri, 163
- variabile, 164
- controale casetă cu listă creare, 118-119
- mesaje de notificare, 130-131
- populare, 128-130
- subclase arbore, 117
- variabile, 119
- controale etichetă statică
- adăugare, 95-97
- formatarea textului, 94
- identificator, 96
- operații la momentul execuției, 95-100
- variabile, 97
- controale glisor, 149
- adăugare în casetele de dialog, 150-151
- canale rectangulare, 152
- Fire, aplicație, 141-142
- gradații, 152-153
- inițializare, 152
- mesaje de notificare, 153-154
- poziție, 152
- proprietăți, 151
- urmărirea deplasării, 153-154
- valori de incrementare/decrementare, 152
- variabile, 151-152
- controale indicator de evoluție, 137
- adăugare în casetele de dialog, 137-139
- Fire, aplicație, 141-142
- interval, 140
- manipulare, 140
- poziție, 141-142
- stiluri, 138
- valoare de increment, 142
- variabile, 139-140
- controale listă
- creare, 119-121
- populare, 132-136
- prezentare, 115
- reprezentare, 120
- selectarea liniilor, 432
- copiere/lipire, 138
- identificatori, 138
- linii de ghidare, 66-67
- liste de imagini, 238-243
- orientate spre intervale, 137
- selector de dată/oră, 154
- adăugare în casetele de dialog, 155-156
- afișare, 158
- determinarea valorilor, 160
- formate de dată/oră, 157-159
- inițializare, 157-160
- interval, 157
- manipularea obiectelor, 167
- mesaje de notificare, 160-162
- proprietățile calendarului, 160
- rata de derulare, 164-165
- stiluri, 155-156
- validitate, 157
- variabile, mapare, 156
- subclasare
- clase ActiveX, 609
- înregistrare, 627
- licențiere, 628
- math.h, fișier de antet, 611
- OLE\_COLOR, tip de date, 618
- pagini de proprietăți, 622
- proprietăți, 617, 621
- tratarea mesajelor, 607
- controale de editare derulabile, 409-410
- controale standard, 140
- controlată, execuție (depanare), 639
- opțiuni de execuție, 646-647
- puncte de întrerupere
- comutare, 645
- stabilire, 644
- Controls, bară cu instrumente (Developer Studio), 60, 62-64
- Controls, paletă
- adăugarea controalelor ActiveX, 189
- în cadrul editorului de dialoguri, 190
- testarea controalelor ActiveX
- în cadrul editorului de dialoguri, 191
- convenții de denumiri
- casete de dialog, 57
- clase (controale ActiveX), 606-608
- clase driver de dispecer, 599
- COM, 582, 599
- variabile, 42

- coolbar. *Vedeți* bare suport
- coordonate
- determinare, 359
- penițe pentru desenare, 358-360
- proporția dimensiunilor, 512
- spațiu de coordonate, 337-339
- TextOut(), parametrii funcției, 379
- copiere, 558-560
- CountClipboardFormats(), funcție, 557
- CPaintDC, clasă, 330
- CPen, clasă, 349-350
- CPoint, clasă, 358
- cpp, extensie de fișier, 50, 536
- CPreviewDC, clasă, 518
- CPrintDialog, clasă, 521
- CPrintInfo(), funcție, 508-509
- CPropertyPage, clasă, 204
- CPropertySheet, clasă, 204
- crearea aplicațiilor. *Vedeți* AppWizard
- Create New Data Source, casetă de dialog, 568
- Create(), funcție, 106, 196-198, 461-462
- casete de dialog
- nemodale, 213
- CDialogBar, obiect, 305
- CStatusBar, obiect, 310
- CreateActiveView(), funcție, 474
- CreateBrushIndirect(), funcție, 372
- CreateClipper(), funcție, 667
- CreateCompatibleBitmap(), funcție, 336
- CreateCompatibleDC(), funcție, 335
- CreateDispatch(), funcție, 600
- CreateEx(), funcție, 294, 310
- CreateFont(), funcție, 387-392
- CreateFontIndirect(), funcție, 395
- CreateHatchBrush(), funcție, 372, 524
- CreateInstance(), funcție, 587
- CreateInstanceLicense(), funcție, 628
- CreateMenu(), funcție, 285
- CreateNewDocument(), funcție, 488
- CreateNewFrame(), funcție, 488
- CreatePalette(), funcție, 667
- CreatePatternBrush(), funcție, 372
- CreatePen(), funcție, 353-354
- CreatePointFont(), funcție, 386
- CreatePrinterDC(), funcție, 526
- CreateSolidBrush(), funcție, 372
- CreateSoundBuffer(), funcție, 657
- CreateStatic(), funcție, 466
- CreateStockObject(), funcție, 351
- CreateSurface(), funcție, 667, 671
- CreateSysColorBrush(), funcție, 370-372
- CreateView(), funcție, 466
- CRecentFileList, clasă, 545-546
- CRecordset, clasă, 571
- actualizarea valorilor, 575
- depanare, 575
- interogări, 572-575
- schimb de câmpuri, 576
- CRecordView, clasă, 249, 576-577
- CRect, clasă, 178, 359
- CRectTracker, clasă, 181
- cadru elastic, 182-184
- indicatori de stil, 182
- personalizare, 183
- CRichEditView, clasă, 249, 450
- CRotaryCtrl, clasă, 606
- CScrollView, clasă, 249
- CSDICoinApp, clasă, 250
- CSDICoinDoc, clasă, 250
- CSDICoinView, clasă, 251
- CShapesDlg, clasă, 240-241
- CSimpTextView, clasă, 380
- CSplitterWnd, clasă, 458-459
- CStatic, funcțiile clasei, 231-232
- CStatusBar, obiect, 310
- CString, clasă, 102-103, 591-592
- CStringList, clasă, 130, 427-428
- CToolBar, obiect, 293, 299-300
- CTreeCtrl(), funcție, 469
- CTreeCtrl, clasă, 118, 125
- CTreeView, clasă, 249, 468-469
- culoare
- bare de dialog, 308-309
- bitmap-uri, 228-229, 367-369
- imprimante monocrome, 524
- limitări hardware, 350
- modificare dinamică, 667

pagină pentru  
 proprietățile generice de  
 culoare, 619  
 penițe, 350  
 pensule, 365  
     *bitmap-uri*, 367-369  
     *contexte dispozitiv*,  
     371-372  
     *fundal*, 367  
     *modele*, 367-369  
     *pensule cu culori*  
     *sistem*, 369-371  
     *ștergere*, 372  
 text, 382  
 curbe cubice, 362-369  
 curbe polinomiale, 362-369  
 curbe, trasare, 362-369  
 cursive, 388  
 cursoare, 46  
     proiectare, 223  
     punct senzitiv, 180  
     test de clic, 180-181  
 CUserException (funcție  
 DDV), 211  
 CUsingDBSet, clasă,  
 572-573  
 Custom Color Selector,  
 clasă, 229  
 Customize, casetă de  
 dialog, 61-62  
 Customize, comandă  
 (meniul Tools), 223  
 CView, clasă, 249, 508-509  
 CWinApp, clasă, 480  
 CWnd, clasă, 77  
     BeginPaint(), funcție,  
     334  
     RedrawWindow(),  
     funcție, 330  
     SetWindowText(),  
     funcție, 78

## D

Database Options, casetă  
 de dialog, 569-570  
 date  
     accesare, 261-262  
     aplicații MDI  
         *accesare*, 492-493  
         *modificare*, 493-496  
         *stocare*, 490-492  
     casete de dialog  
         *schimb*, 207-208  
         *validare*, 213  
     încapsulare, 248  
     manipulare, 539-543  
     obiecte  
         *serializabile*,  
         533-536, 539  
         *serializare*, 543-545  
     stocare, 261-262  
     surse, 566-568  
     tipuri de date  
         BCHAR, 396  
         OLE\_COLOR,  
         subclasare, 618  
         POINT, 414  
         vectori, 590-529  
     transfer, 556-562  
     validare  
         DDV, 210-211  
         DDX, 207-208, 210  
         specifică, 212-213  
 Date Time Picker  
     Properties, casetă de  
     dialog, 155  
 Date/Reprezentare,  
 arhitectură, 246, 260-265,  
 267-268  
 DCL (limbaj pentru  
 controlul datelor),  
 565-566  
 DCOM (COM distribuit),  
 587  
 DDE (schimb dinamic de  
 date), 556-558  
 DDL (limbaj pentru  
 definirea datelor),  
 565-566

DDV  
     funcții, 210-212  
     funcții proprii, 212  
 DDX (schimb de date),  
 207-208  
     Text(), funcție, 208  
     validarea datelor,  
     207-208, 210  
 deblocarea bufferelor, 658  
 Debug Assertion Failed,  
 casetă de dialog, 642  
 Debug Info, 634  
 Debug, configurație,  
 632-633  
 declararea claselor  
 serializabile, 534-536  
 DECLARE\_DYNCREAT  
 E, macrodefiniție,  
 254-255, 485  
 decupare  
     contexte dispozitiv  
     pentru redesenare, 331  
     ferestre derulante,  
     421-422  
     text în dreptunghiuri,  
     384-385  
 decupare, liste  
 (DirectDraw), 667  
 Deflate(), funcție, 375  
 DefWindowProc(), funcție,  
 111  
 DeleteAllItems(), funcție,  
 127  
 DeleteBlobs(), funcție, 540  
 DeleteContents(), funcție,  
 259  
 DeleteItem(), funcție, 125,  
 443  
 DeleteObject(), funcție,  
 353-354, 402  
 depanare  
     contexte dispozitiv  
     pentru redesenare, 331  
     ferestre derulante,  
     421-422

text în dreptunghiuri,  
 384-385  
 depanare, versiune, 24-25  
     ASSERT, macrodefiniție,  
     642-644  
     copiere pe blocuri,  
     funcții, 667-668  
     CRecordset, funcțiile  
     clasei, 575  
     depanare, configurație,  
     632-633  
     diagnosticare,  
     instrucțiuni, 639  
     excepții generate de  
     bazele de date, 575  
     imediată, 638  
     în rețea, 639  
     la distanță, 638  
     MFC Tracer, instrument,  
     653  
     OLE/COM Object  
     Viewer, instrument, 653  
     opțiuni/nivele  
         Generate Browse  
         Info, 634  
         informații pentru  
         depanare, opțiuni,  
         635  
         nivele de avertizare,  
         633  
         Preprocessor  
         Definitions, 633-634  
         Project Options, 634  
         Warnings as Errors,  
         633  
     pas cu pas, 639  
         editare codului, 647  
         opțiuni de execuție,  
         646-647  
         puncte de întrerupere,  
         644-645  
     Process Viewer,  
     instrument, 652  
     proiecte  
         algoritmi de sortare,  
         642

    combinatii de taste,  
     640  
     indicatori pentru  
     compilare, 635  
     puncte de întrerupere,  
     condiții, 639  
     puncte de întrerupere,  
     639  
     Source Browser,  
     instrument  
         clase de bază,  
         opțiune, 636  
         clase derivate,  
         opțiune, 637  
         definiții și referințe,  
         opțiune, 636  
         identificatori,  
         introducere, 636  
         lista apelanților,  
         opțiune, 637  
         lista apelanților,  
         opțiune, 637  
         sumar de fișier,  
         opțiune, 636  
     Spy++, instrument,  
     650-653  
         Messages,  
         reprezentare,  
         650-651  
         Processes,  
         reprezentare, 652  
         Threads,  
         reprezentare, 652  
         Windows,  
         reprezentare, 651  
     stiva de apeluri, afișare,  
     647-648  
     TRACE, macrodefiniții,  
     639-642  
         *listing*, 640-641  
         *rezultat*, 642  
     variabile, 647-649  
     VERIFY, macrodefiniție,  
     642-644  
 depanator  
     Disassembly, bara cu  
     instrumente, 649

editare și continuare,  
 opțiune, 647  
 execuție controlată,  
 întârzieri, 646  
 ferestre  
     Call Stack, fereastră,  
     647-648  
     Disassembly,  
     fereastră, 649  
     Memory, fereastră,  
     649  
     QuickWatch,  
     fereastră, 649  
     Registers, fereastră,  
     649  
     Variables, fereastră,  
     647-649  
     Watch, fereastră,  
     647-649  
 derivate, clase, 41  
 derulante, liste, 114-115  
 derulare  
     ferestre  
         dimensiuni de  
         derulare, 416-418  
         pagină/linie,  
         incrementale de  
         derulare, 418-420  
         tratarea mesajelor,  
         421-422, 424  
     selector de dată/oră,  
     control, 164-165  
     valori de derulare, 419  
 deschidere  
     fișiere, 547-548  
     proiecte, 36  
 deschiderea fișierelor,  
 moduri (CFile), 548  
 desenare  
     CDC, clasă, 324  
     contexte dispozitiv  
         selectare, 355  
         spațiu de coordonate,  
         337-339  
     tipuri, 323

- controale ActiveX, 609-612
- indicatori de caracteristici, 343
- moduri de mapare, 337-339
- penițe, 355
  - cercuri*, 360-362
  - curbe*, 362-369
  - dreptunghiuri*, 359-360
  - elipse*, 360-362
  - figuri umplute*, 349
  - linii*, 357-358
  - NULL\_PEN*, stil, 351
  - poligoane*, 364-365
  - punct de coordonate*, 358-360
- pensule
  - cercuri*, 374-375
  - dreptunghiuri*, 372-374
  - elipse*, 374-375
  - poligoane*, 375-377
  - sectoare*, 374-375
  - suprafețe de coardă*, 374-375
- poziție, determinare, 356
- reprezentări
  - datele documentului*, 262-264
  - redesenare*, 175
- desfășurarea pe rânduri a textului, 402
- DestroyWindow(), funcție, 195, 218
- DetachDispatch(), funcție, 601
- determinare. *Vedeți accesare*
- Developer Studio
  - bară de titlu, 39
  - casete de dialog
    - alinie*, 66
    - aspect*, 60, 62-64
  - dimensionare*, 64
  - editare*, 65
  - grupare*, 68-69
  - linii de ghidare*, 66-67
  - ordine de selectare*, 69-70
  - taste de acces*, 70-71
- casete de dialog, controale
  - denumire*, 58-60
  - dimensionare*, 62-63
  - proiectare*, 54-58
  - proprietăți*, 58
  - stiluri*, 59-60
- Controls, bară cu instrumente, 60, 62-64
- Dialog, bară cu instrumente, 60, 62-64
- Layout, meniu, 60
- Output, fereastră, 36
- Project Workspace, fereastră
  - ClassView*, 38-44
  - FileView*, 48-49
  - pagini*, 38
  - ResourceView*, 45-48
- rolare în fundal, 592
- DevMode, structură, 520
- dezactivare
  - butoane
    - bare cu instrumente*, 299-300
    - dezactivat, stare*, 235-237
  - comenzi de meniu, 278
  - mnemonice, 95
- diagnosticare, instrucțiuni, 639
- Dialog Editor, 190
- Dialog Properties, casetă de dialog, 58
- dialog, bare, 333. *Vedeți și bare cu instrumente*
- alinie, 306
- ancorare*, 306
- butoane, 304
- controale, 304, 307-308
- culoare, 308-309
- editare, marcaje de ghidare, 305
- ferestre cadru, 305-306
- funcții, 307-308
- Internet Explorer, bare suport, 319-321
- machete, 304
- redesenare, 309
- resurse, 304
- Dialog, bară cu instrumente (Developer Studio), 60, 62-64
- dialog, casete
  - About, 607
  - Add Event, 615
  - Add Member Function, 31, 41-43, 80, 266
  - Add Member Variable, 41, 74, 80, 84-85, 108
  - Add Property, 617
  - Adding a Class, 202
  - adăugarea controalelor ActiveX, 189
  - Ambient Properties, 629
  - Bitmap Properties, 228
  - butoane
    - buton implicit*, 63
    - supradefinirea funcțiilor*, 48
  - Buttons, proiect, 72-74
  - calendar, control, 162-164
  - casetă cu listă, control
    - creare*, 118-119
    - mesaje de notificare*, 130-131
    - populare*, 128-130
    - subclase arbore*, 118
    - variabile*, 119
  - casete combinate
    - creare*, 115

- derulare*, 114-115
- determinarea conținutului*, 124-125
- imagini*, 116
- mesaje de notificare*, 124-125
- populare*, 121-124
- sortare*, 115-116
- variabile*, 115-116
- casete de editare
  - adăugare*, 100-102
  - determinarea textului*, 102-104
  - distincția minuscule/majuscule*, 100
  - inițializarea câmpurilor*, 102-104
  - mesaje de notificare*, 104-105, 107
  - moștenirea claselor*, 107-112
  - multi-linie*, 112
  - numere*, 100
  - parole*, 100
  - taste mnemonice*, 95, 102
  - testare*, 102
  - tratarea mesajelor*, 104-105
  - variabile*, 102-103
- casete de grupare, 68, 83
- casete de validare
  - stare*, 89-92
  - tri-stare*, 88
- cataloge de proprietăți, 204
- Choose Font, 397-398
- CityBreak, proiect, 83-88
- Class, 284
- ClassWizard, 84-85, 233, 262, 464-465
- Combo Box Properties, 114-115
- Components and Controls Gallery, 460
- Confirm Classes, 188
- controale
  - accesare*, 69
  - alinie*, 66
  - casete de editare, controale*, 95
  - casete de validare*, 69
  - dimensionare*, 64
  - dispunere*, 60, 62-64
  - editare*, 65
  - grupare*, 68-69
  - linii de ghidare*, 66-67
  - localizare*, 154
  - ordinea de selectare*, 69-70
  - taste de acces*, 70-71, 95
- Create New Data Source, 568
- Custom Color Selector, 229
- Customize, 61-62
- Database Options, 569-570
- date
  - schimb*, 207-208
  - validare*, 213
- Date Time Picker Properties, 155
- DDV, 210-211
- DDX, 207-208, 210
- Debug Assertion Failed, 642
- denumire, 58-60
- Dialog Properties, 58
- dialog, clase, 200
  - casete de dialog modale*, 204-205
  - CDialog*, 201-203
  - inițializare*, 203-204
  - resursă machetă de dialog nouă*, 201
  - variabile membru*, 206-207
- Edit Names, 606
- Edits, 100-114
- erori, 124
- Event Log, 630
- gestionare, 206
- Group Box Properties, 83
- Icon Properties, 226-227
- IDD\_ABOUTBOX, 56-57
- IDD\_PERSON\_TEMPL\_ATE, 56-57
- Images, 233
- imagine, control
  - adăugare*, 232
  - încărcare la momentul execuției*, 231
  - tipuri de imagini*, 230-231
  - variabile*, 233, 235
- Import Resource, 229
- indicator de evoluție, control, 137-139
- Insert ActiveX Control, 190
- Insert Object, 607
- Insert Resource, 45, 201, 226-227
- Lists, 114-115
- listă, control
  - creare*, 119-121
  - populare*, 132-136
  - prezentare*, 115
- Menu Item Properties, 265, 272, 494
- Message Options, 650-651
- MFC Assert, 474-475
- MFC ClassWizard, 240
- modale, 200, 205
- Multiple Selection Properties, 73
- nemodale, 200, 213
  - creare/distrugere*, 213-216

închidere, 218  
 mesaj de închidere, 220  
 opțiune de închidere, 221  
 New Class, 108, 464-465  
 New Document, 485  
 New Icon Image, 226  
 New Project, 54  
 New Project Information, 23, 248, 478  
 New Windows Message and Event Handlers, 170  
 ODBC Data Source Administrator, 566  
 ODBC Microsoft Access Setup, 567  
 Picture Properties, 231-232  
 Print, 519-522  
     *accesare directă*, 526-528  
     *Collate, casetă de validare*, 519  
     *scurt-circuitare*, 516  
 Progress Properties, 138  
 proiectare, 54-58  
 Project Settings, 51, 632  
 proprietăți, 58  
 puncte de întrerupere, 644  
 selector de dată/oră, control, 155-156  
 dialog, clase, 200  
     afișarea casetelor de dialog modale, 204-205  
     CDialog, 201-203  
     inițializare, 203-204  
     resursă machetă de dialog nouă, 201  
     variabile membru, 206-207  
 dialog, tip de aplicație, 532  
 dialog, unități (DLU), 59

**dimensionare**  
     bare cu instrumente, 294  
     bitmap-uri, 228-229  
     casete de dialog, 62-64  
     evenimente, 405, 407-408  
**dimensionare, opțiuni (fonturi), 390**  
**dimensiuni de pagină, 523**  
**dinamice, ferestre multi-sectiune**  
     creare, 458-459  
     inițializare, 461-463  
**dinamice, mulțimi de înregistrări, 569-570**  
**dinamice, reprezentări, 470-475**  
**Direct3D, 673-674**  
**DirectDraw**  
     accesare, 667  
     agregare, 666  
     comutare, 667-668  
     crearea obiectelor, 666  
**DirectDrawCreate(), funcție, 666**  
     dublu buffer, 667-668, 671-672  
     GDI, funcții, 667  
     HAL, 665  
     HEL, 665  
     liste cu informații pentru decupare, 667  
     ModeX, 665  
     modificarea dinamică a culorilor, 667  
     nivele de cooperare, 667  
     palette, 667  
     suprafețe, 667-668  
**directe, obiecte (Direct3D), 673**  
**DirectInput, 674**  
**DirectPlay, 674**  
**DirectSetup, 674**  
**DirectSound**  
     blocare/deblocare, 658

buffere  
     *mixare*, 658-659, 662-664  
     *primar*, 657  
     *redare*, 658  
     *repetare*, 658  
     *secundar*, 657-658  
     *semnal de undă*, 655-656  
**DirectSoundCreate(), funcție, 657**  
**DirectSoundEnumerate(), funcție, 657**  
     interfață, 656-657  
     nivele de cooperare, 657, 663  
     zgomot alb, 657  
**DirectX**  
     extensii, 674  
     Game SDK, 655-656  
**DirectXRegisterApplication(), funcție, 674**  
**DirectXSetup(), funcție, 674**  
**Disassembly, bară cu instrumente (depanator), 649**  
**Disassembly, fereastră, 649**  
**disc, fișiere, 551**  
**dispecer, clase, 188**  
     clasă driver de dispecer  
         *clienți de automatizare*, 599-601  
         *nume*, 599  
     crearea controalelor ActiveX, 196-198  
     funcții, 188-189  
     variabile membru, 193  
**dispecer, interfață (automatizare OLE), 589-590**  
**dispunere**  
     bare de dialog, marcaje de ghidare, 305

casete de dialog, controale, 60, 62-64  
**DISP\_FUNCTION, macrodefiniție, 597**  
**DLL**  
     biblioteci de intermediere, apeluri intermediare, 588  
     grafică, 323  
**DLU (unități de dialog), 59**  
**DML (limbaj pentru manipularea datelor), 565-566**  
**DMS (sistem pentru gestionarea bazelor de date), 564**  
**DockControlBar(), funcție membru, 296**  
**Document/Reprezentare, arhitectură, 246, 260-265, 267-268**  
**documente. Vedeți și MDI; SDI**  
     accesare, 262-264  
     atribute, 553-556  
         *GetFileName()*, funcție, 554  
         *GetStatus()*, funcție, 553  
     căutare, 127  
     citire, 548-551  
     clase document  
         *funcții membru publice*, 481  
         *stocarea datelor*, 490-492  
         *variabile membru protejate*, 261-262  
     ClassWizard, regenerare, 50  
     conținut (control de editare), 550  
     date  
         *accesare*, 261-262  
         *manipulare*, 539-543  
         *stocare*, 261-262

definire, 246  
 deschidere, 547-548  
 documente active, 481, 602  
 fișiere compuse, 589, 595-596  
     *moniker*, 603  
     *serializare*, 602  
 fișiere de antet, definiții de clase, 534  
 fișiere de asistență (controale ActiveX), 605-606  
 fișiere de resurse, 56-57  
 fișiere sursă (controale ActiveX), 627  
 importarea resurselor, 229-230  
 indicatori de la sistemul de operare, 546  
 înregistrarea tipurilor de documente, 545-546  
 lista documentelor folosite recent, 545-546  
 liste, 127  
 lungime, 553  
 Moniker, 589  
 nume de fișiere, 550  
     *extensii*, 50  
     *redenumire*, 555  
 poziție, 548-549  
     *GetPosition()*, funcție, 552  
     *manipulare*, 552  
 reprezentare, 490  
 scriere, 548-551  
 serializare (WordPad), 603  
 stare, 553  
 ștergere, 50, 555  
 tipărire, 515-518  
 tratare, 546-556  
**DoDataExchange(), funcție, 207-210**  
**DoFieldExchange(), funcție, 576-577**  
**DoModal(), funcție, 213**  
**DoPreparePrinting, funcție, 516**  
**DoPropExchange(), funcție, 625**  
**DoVerb(), funcție, 603**  
**DPTOLP() funcție, 339**  
**DrawIcon(), funcție, 171**  
**DrawText(), funcție, 400-401**  
**DrawTextEx(), funcție, 400**  
**DrawTrackerRect(), funcție, 183**  
**dreptunghiuri**  
     controale imagine, 231  
     desenare  
         *penițe*, 359-360  
         *pensule*, 372-374  
     invalidare, 421-422  
     suport pentru tipărire, 511  
     text, 384-385  
**drivere, 504**  
**DRIVERVERSION, indicator, 343**  
 dsp, extensie de fișier, 50  
 dsw, extensie de fișier, 50  
 duale, interfețe, 590  
 dublu buffer (DirectDraw), 667-668, 671-672  
 dublu clic, eveniment, 174

**E**

ecrane. *Vedeți API; interfețe; reprezentări; ferestre*  
**Edit Names, casetă de dialog, 606**  
**Edit, meniu, comanda Set Ambient Properties, 628**  
**editare**  
     bară de dialog, marcaje de ghidare, 305  
     casetă de dialog, controale, 65

elemente de meniu, 289-290  
 pictograme, 225  
**editare formatată, tip de reprezentare, 450**  
 formatarea paragrafelor, 451, 453  
 încărcarea/salvarea textului, 451  
 obiecte OLE, inserare, 453  
**editor de text, 411**  
**editor, editor de resurse, 297-299**  
**editor, tip de reprezentare, 450**  
**editorul de resurse. *Vedeți* resurse**  
**Edits, casetă de dialog, 100-114**  
**elaborate, obiecte (Direct3D), 673**  
**elemente de meniu**  
 inserare, 271-272  
 marcaje de validare, 274-275  
 submeniuri, 274  
**elipse**  
 penițe, 360-362  
 pensule, 374-375  
**Ellipse(), funcție, 264, 359-360, 374, 420**  
**Enable(), funcție, 278**  
**EnableDocking(), funcție, 295**  
**EnableMenuItem(), funcție, 289**  
**EnableScrollBar(), funcție, 144**  
**EndDialog(), funcție, 205**  
**EndDoc(), funcție, 528**  
**EndPage, funcție, 529-530**  
**EnumClipboardFormats(), funcție, 557**  
**enumerarea fonturilor, 393-398**  
**EnumFontFamilies(), funcție, 393**  
**ENUMLOGFONT, structură, 396-397**  
**erori (casete de dialog), 124**  
**Escape(), funcție, 529-530**  
**escape, coduri (specifice dispozitivelor), 529-530**  
**etichetă statică, control, 94**  
 adăugare, 95-97  
 casete de editare, 95  
 formatarea textului, 94  
 identificator, 96  
 operații la momentul execuției, 95-100  
 variabile, 97  
**ETO\_CLIPPED, indicator, 384-385**  
**evenimente**  
 browser HTML, tip de reprezentare, 455  
 controale ActiveX  
 adăugare, 615  
 generare, 614-617  
 jurnalizare, 630  
 tratare, 612-614  
 derulare, 421-422, 424  
 ferestre  
 cadre, 252  
 dimensionare, 405, 408-409, 412  
 reprezentare, 650-651  
 mouse, captură, 179  
 mouse, deplasare  
 parametri, 175  
 poziția cursorului, listing, 175-176  
 urmărirea cursorului, listing, 177-178  
 variabile membru, 176  
 notificare  
 casete combinate, 124-125

*casete de editare, 104-105, 107*  
 punct senzitiv, cursor, 180  
 repere orare, 169  
 Repositioned, 615  
 tratare. *Vedeți și* funcții de tratare  
 adăugare cu  
 WizardBar, 413-414  
 BN\_CLICKED, 80  
 casete de editare, 104-105  
 clic asupra butonului, eveniment, 74, 169-173  
 controale ActiveX, 196-199, 612-614  
 GetNextItem(), funcție, 439  
 mouse, evenimente generate de butoane, 169-174  
 WM\_DESTROY, 328  
 WM\_ERASEBKGD, 334  
 WM\_MOUSEMOVE, 175  
 WM\_PAINT, 330  
**evenimente generate (controale ActiveX), 614-617, 630**  
**Event Log, casetă de dialog, 630**  
**excepții, depanare, 575, 632**  
**ExchangeVersion(), funcție, 625**  
**executabile, fișiere, 229**  
**execuție controlată, operații (depanare), 639**  
 opțiuni de execuție, 646-647  
 puncte de întrerupere  
 comutare, 645  
 stabilire, 644

**ExecWB(), funcție, 454**  
**explicații flotante**  
 afișare, 294  
 stiluri, 310  
**Explorer**  
 bară cu instrumente, 294  
 bare suport  
 ancorare, 319  
 AppWizard, 319-321  
 bitmap de fundal, 319-321  
 controale încapsulate, 320  
 stiluri, 319-321  
 titlu, 321  
 browser HTML, tip de reprezentare, 454  
**expresii (QuickWatch), 649**  
**extensii**  
 bare cu instrumente, 297  
 DirectX, 674  
 Edits, casetă de dialog, 108  
**ExtTextOut(), funcție, 379, 384-385**

## F

**familie (de fonturi), opțiuni, 390**  
**faxuri**  
 Mail and Fax, panoul de control, 675-676  
 tipărire, proporția dimensiunilor, 512  
**fErase, indicator, 334**  
**ferestre. *Vedeți și* API; interfețe; reprezentări**  
 actualizare, 330  
 adăugarea de reprezentări, 470  
 butoane, 76  
 cadre  
 ancorare, 293  
 bare cu instrumente, 301-302  
 bare cu instrumente ancorabile, 294-296  
 bare de dialog, 305-306  
 mesaje, 252  
 capturarea mouse-ului, 179  
 contexte dispozitiv (clasa CDC), 324  
 crearea la momentul execuției, 461-462  
 CWnd, clasă, 77  
 depanator  
 Call Stack, 649  
 Memory, 649  
 QuickWatch, 649  
 Registers, 649  
 Variables, 647-649  
 Watch, 647-649  
 derulare  
 dimensiuni de derulare, 416-418  
 mesaje de derulare, 421-422, 424  
 pagină/linie, incremente de derulare, 418-420  
 dinamice, ferestre multi-secțiune  
 creare, 458-459  
 inițializare, 461-463  
 dinamice, reprezentări, 470-475  
 Disassembly, 649  
 ferestre multi-secțiune, prezentare, 458  
 poziție, 404  
 redesenare, 175  
 redimensionare  
 casete de dialog dimensionabile, 415  
 dimensionare, eveniment, 405, 407-408  
 fixarea dimensiunilor, eveniment, 408-409, 412  
 formular, tip de reprezentare, 404  
 limitări asupra dimensiunilor, 414-415  
 SDI, elementele aplicației, 251-253  
 secționare, 458-459  
 spațiul de lucru al proiectului  
 ClassView, 38-44  
 FileView, 48-49  
 pagini, 38  
 ResourceView, 45-48  
 standard, bară cu instrumente, 294  
 stare, 404  
 statice, ferestre multi-secțiune  
 creare, 464  
 crearea aplicațiilor pentru gestionarea fișierelor, 468-469  
 inițializare, 464-467  
**figuri, desenare, 349**  
**File, meniu, comanda Open As Resources, 229**  
**FileView (fereastra spațiului de lucru al proiectului), 48-49**  
**FillRect(), funcție, 372**  
**filtre (infrastructură SDI), 533**  
**FindFirstFile(), funcție, 127**  
**FindNext(), funcție, 680**  
**FindNextFile(), funcție, 127**  
**fire de execuție, 652**  
**Fire, controalele aplicației, 141-142**  
**FireRepositioned(), funcție, 615**



fixarea dimensiunilor, eveniment, 408-409, 412  
 fișiere. *Vedeți* documente; MDI; SDI  
 fișiere de asamblare, 51  
 Flip(), funcție, 667-668, 672  
 FloatControlBar(), funcție, 301  
 FloatControlBar(), funcție membru, 297  
 flotantă, bară cu instrumente  
   comentarii în cod, 295  
   dimensionare, 294  
   inserare, 301  
 FmtLines(), funcție, 112  
 focus (reprezentare), 300  
 focus, stare (butoane bitmap), 235-237  
 FontCallback(), funcție, 396  
 fonturi. *Vedeți* text  
 Format(), funcție, 98-99, 158  
 formatare  
   afișare, 158  
   Clipboard, 556-558  
   text, 94  
 formular, tip de reprezentare, 404  
 FreeLibrary(), funcție, 681  
 frunză, nod, 116  
 funcții  
   AbortDoc(), 525-526  
   acces, 510  
   AddDocTemplate(), 485  
   AddRef(), 583  
   AddString(), 129, 219  
   AddView(), 470  
   adăugare, 41-43  
   AfxGetApp(), 489  
   AfxGetMainWnd(), 302, 489  
   AfxMessageBox(), 200

AfxOleInit(), 560, 595-596  
 AllocSysString(), 591-592  
 AmbientBackColor(), 611  
 AmbientForeColor(), 611  
 AppendMenu(), 286-287  
 ApplyColor(), 308  
 ASSERT\_KINDOF(), 474-475  
 Attach(), 286  
 AttachClipboard(), 562  
 AttachDispatch(), 601  
 automatizare, servere, 597-599  
 bare de dialog, 307-308  
 BeginPaint(), 334  
 BitBlt(), 335-336, 667-668  
 Blt(), 667, 672  
 CBitmap(), 369  
 CDialog, OnOK(), 56-57  
 CenterPoint(), 358, 420  
 CFrameWnd(), 489  
 ChangeDialogTitle(), 79  
 CheckMenuItem(), 289  
 Chord(), 375  
 ClearTics(), 153  
 Close(), 550  
 CLSIDFromString(), 592  
 CModeless(), 214  
 CMouseMsgDlg(), 181  
 CoCreateInstance(), 587, 592, 666  
 CoCreateInstanceEx(), 587  
 CoGetObject(), 587  
 CoInitialize(), 671  
 ColorChange(), 309  
 context dispozitiv, 525  
 CountClipboardFormats(), 557  
 CPrintInfo(), 508-509

Create(), 106, 196-198, 461-462  
 CreateActiveView(), 474  
 CreateBrushIndirect(), 372  
 CreateClipper(), 667  
 CreateCompatibleBitmap(), 336  
 CreateCompatibleDC(), 335  
 CreateDispatch(), 600  
 CreateEx(), 294, 310  
 CreateFont(), 387-392  
 CreateFontIndirect(), 395  
 CreateHatchBrush(), 372, 524  
 CreateInstance(), 587  
 CreateInstanceLicense(), 628  
 CreateMenu(), 285  
 CreateNewDocument(), 488  
 CreateNewFrame(), 258, 488  
 CreatePalette(), 667  
 CreatePatternBrush(), 372  
 CreatePen(), 353-354  
 CreatePointFont(), 386  
 CreatePrinterDC(), 526  
 CreateSolidBrush(), 372  
 CreateSoundBuffer(), 657  
 CreateStatic(), 466  
 CreateStockObject(), 351  
 CreateSurface(), 667, 671  
 CreateSysColorBrush(), 370-372  
 CreateView(), 466  
 CTreeCtrl(), 469  
 DDV, 210  
 DDX, 208  
 Deflate(), 375  
 DefWindowProc(), 111  
 DeleteAllItems(), 127

DeleteBlobs(), 540  
 DeleteContents(), 259  
 DeleteItem(), 125  
 DeleteObject(), 353-354, 402  
 DestroyWindow(), 195, 218  
 DetachDispatch(), 601  
 DirectDrawCreate(), 666  
 DirectSoundCreate(), 657  
 DirectSoundEnumerate(), 657  
 DirectXRegisterApplication(), 674  
 DirectXSetup(), 674  
 dispecer, 188-189  
 DockControlBar(), 296  
 DoDataExchange(), 207-210  
 DoFieldExchange(), 576-577  
 DoPreparePrinting(), 516  
 DoPropExchange(), 625  
 DoVerb(), 603  
 DPtoLP(), 339  
 DrawIcon(), 171  
 DrawText(), 400-401  
 DrawTextEx(), 400  
 DrawTrackerRect(), 183  
 Ellipse(), 264, 359-360, 374, 420  
 Enable(), 278  
 EnableMenuItem(), 289  
 EnableScrollBar(), 144  
 EndDialog(), 205  
 EndDoc(), 528  
 EndPage(), 529-530  
 EnumClipboardFormats(), 557  
 EnumFontFamilies(), 393  
 Escape(), 529-530  
 ExchangeVersion(), 625-626  
 ExecWB(), 454

ExtTextOut(), 379, 384-385  
 FillRect(), 372  
 FindFirstFile(), 127  
 FindNext(), 680  
 FindNextFile(), 127  
 FireRepositioned(), 615  
 Flip(), 667-668, 672  
 FloatControlBar(), 297, 301  
 FmtLines(), 112  
 FontCallback(), 396  
 Format(), 98-99, 158  
 FreeLibrary(), 681  
 generator de clase, 254-255  
 GetActiveDocument(), 489  
 GetActiveView(), 474-475  
 GetAttachedSurface(), 671  
 GetBackColor(), 188-189  
 GetBkColor(), 382  
 GetChannelRect(), 152  
 GetClientRect(), 178, 359, 420  
 GetComputerName(), 98-99  
 GetCopies(), 521  
 GetCurrentDirectory(), 124  
 GetCurrentPosition(), 356-358, 397  
 GetCurSel(), 124  
 GetDatabase(), 571  
 GetDC(), 667-668, 672  
 GetDefaultConnect(), 574  
 GetDefaultSQL(), 574  
 GetDesktopWindow(), 327  
 GetDeviceCaps(), 342-344, 512  
 GetDlgCtrlID(), 154

GetDlgItem(), 72, 76, 106, 154  
 GetDocument(), 262-264, 489  
 GetDriverName(), 522  
 GetEditControl(), 448  
 GetFileName(), 554  
 GetFirstViewPosition(), 474-475, 489  
 GetFromPage(), 521  
 GetHeadPosition(), 431  
 GetIDsOfNames(), 590  
 GetLBText(), 125  
 GetLine(), 112  
 GetLineCount(), 112  
 GetLineSize(), 152  
 GetMonitorFrequency(), 667-668  
 GetMonthDelta(), 164-165  
 GetNextItem(), 438  
 GetNextView(), 474-475, 489  
 GetNumTicks(), 153  
 GetPageSize(), 152  
 GetParentFrame(), 489  
 GetPolyFillMode(), 375-376  
 GetPortName(), 521  
 GetPrinterDC(), 522  
 GetProcAddress(), 675-676, 680  
 GetRange(), 165  
   *calendar, control, 164-165*  
   *indicator de evoluție, control, 140*  
   *selector de dată/oră, control, 157*  
 GetSafeHwnd(), 412  
 GetScanLine(), 667-668  
 GetScrollInfo(), 145  
 GetScrollLimit(), 424  
 GetScrollPos(), 145  
 GetScrollPosition(), 420

GetSel(), 106  
 GetSelection(), 153  
 GetStatus(), 553  
 GetSystemDirectory(), 124  
 GetTextAlign(), 380  
 GetTextColor(), 382  
 GetTextMetrics(), 463  
 GetTime(), 159  
 GetToday(), 164-165  
 GetToPage(), 521  
 GetTypeInfo(), 590  
 GetTypeInfoCount(), 590  
 GetUserName(), 98-99  
 GetVersionEx(), 98-99  
 GetWindowDC(), 327  
 GetWindowPlacement(), 404  
 GetWindowRect(), 359  
 GetWindowsDirectory(), 124  
 GetWindowText(), 106  
 globale, funcții, 98-99  
 GlobalMemoryStatus(), 98-99  
 Height(), 359  
 HitTest(), 183-184  
 InflateRect(), 362  
 Initialize(), 667, 671  
 InitialUpdateFrame(), 258, 488  
 InitInstance(), 253, 484, 595-596  
 InsertColumn(), 133  
 InsertItem(), 30, 243, 438  
 InsertMenu(), 286-287  
 Invalidate(), 175, 421-422  
 InvalidateControl(), 612  
 InvalidateRect(), 421-422  
 InvalidateRgn(), 421-422  
 Invoke(), 590, 600  
 InvokeHelper(), 188-189, 599  
 IsEmpty(), 104  
 IsIconic(), 176, 331  
 IsKindOf(), 474  
 IsPrinting(), 511  
 IsWindowEnabled(), 80  
 IsWindowVisible(), 78  
 iterator, 393  
 LimitText(), 107  
 LineTo(), 357-358  
 LoadBitmap(), 234-235  
 LoadFrame(), 258, 485, 488  
 LoadLibrary(), 675-676  
 LoadMenu(), 285  
 LoadStandardIcon(), 171-172  
 LoadToolBar(), 294  
 Lock(), 658  
 LockWindowUpdate(), 407-408  
 MakeReverse(), 104  
 MAPI, funcții, 674-677  
 MAPIFreeBuffer(), 681  
 mciGetErrorString(), 688  
 mciGetYieldProc(), 685  
 mciSendCommand(), 682  
 mciSendString(), 682  
 mciSetYieldProc(), 685  
 MCIWndChangeStyles(), 690  
 MCIWndGetMode(), 691  
 MCIWndGetStyles(), 690  
 MDI, acces, 488  
 MDIGetActive(), 488  
 membru, funcții  
     *casete de dialog nemodale*, 216  
     *ClassView*, 80  
     *CListCtrl*, clasă, 132  
     *CListVDoc*, clasă, 428  
     *CPrintDialog*, clasă, 521  
     *create*, 471

*CStatic*, clasă, 231-232  
*funcții membru publice*, 481  
 MessageBox(), 87  
 ModifyMenu(), 289-290  
 MoveTo(), 355-357  
 Navigate2(), 454  
 OffsetPos(), 141-142  
 OLE, automatizare, 589  
 OnActivate(), 481  
 OnActiveView(), 474-475  
 OnBeforeNavigate2(), 455  
 OnBeginPrinting(), 523  
 OnChangedViewList(), 470  
 OnChangeEditSecond(), 105  
 OnChar(), 110  
 OnChoosefont(), 397-398  
 OnClickCalendar(), 196  
 OnClose(), 220  
 OnCloseDocument(), 259, 470  
 OnContextMenu(), 281-283  
 OnCreate(), 293, 461  
 OnCreateClient(), 461  
 OnDoVerb(), 603  
 OnDownloadBegin(), 456  
 OnDownloadComplete(), 456  
 OnDraw(), 263-264, 339, 355-357, 504-508  
 OnDrawit(), 329, 332  
 OnDrawSplitter(), 464-465  
 OnEditCopy(), 558  
 OnEnableDisable(), 78  
 OnEndlabeledit(), 447  
 OnEndPrinting(), 523

OnEraseBkgnd(), 366, 372  
 OnFileNew(), 257-258  
 OnFileReceive(), 680  
 OnGetMinMaxInfo(), 413-414  
 OnGetText(), 103  
 OnHScroll(), 424  
 OnInitDialog(), 132  
 OnInitDialog(), 132, 164-165, 194, 240, 600  
 OnInitialUpdate(), 258, 416, 463, 488, 574  
 OnKeyDown(), 663  
 OnLButtonDown(), 612  
     *capturarea mouse-ului*, 179  
     *indicatori*, 171  
     *listing*, 170-171  
 OnMouseMove(), 612  
     *parametri*, 175  
     *poziția cursorului*, *listing*, 175-176  
     *urmărirea cursorului*, *listing*, 177-178  
     *variabile membru*, 176  
 OnNewDocument(), 259  
 OnOk(), 86  
 OnOpenDocument(), 259  
 OnPaint(), 176  
 OnPreparePrinting(), 515  
 OnPressMe(), 32  
 OnPrint(), 504, 508-510  
 OnProgressChange(), 456  
 OnSaveDocument(), 259  
 OnScroll(), 421-422  
 OnSelchangeSubDirs(), 130  
 OnSelMainDir(), 124  
 OnSetFocusEditShow(), 106  
 OnShowArt(), 471  
 OnShowEdit(), 471

OnShowHide(), 74  
 OnSize(), 405, 411  
 OnSizing(), 406  
 OnStartModeless(), 215  
 OnStopModeless(), 216  
 OnTimer(), 667-668, 672  
 OnUpdate(), 495  
 OnUpdateMousePos(), 313  
 OnUpdateShowArt(), 474  
 OnUpdateShowEdit(), 474  
 Open(), 547, 574  
 OpenDocumentFile, 257-258, 488  
 OpenEx(), 570-572  
 ParseCommandLine(), 255-256, 595-596  
 ParseParam(), 255  
 Play(), 658  
 PlayFreq(), 662  
 PolyBezier(), 362-369  
 PolyBezierTo(), 364-365  
 Polygon(), 375-376  
 Polyline(), 364-365  
 PolyLineTo(), 364-365  
 PolyTextOut(), 379  
 PopulateCombo(), 122  
 PopulateListBox(), 128  
 PopulateTree(), 125  
 PreTranslateMessage(), 169  
 PrintAll(), 522  
 PrintCollate(), 522  
 PrintRange(), 522  
 PrintSelection(), 522  
 ProcessShellCommand(), 256-257, 486  
 proxim, 188-189  
 PtInRect(), 181-182  
 qsort(), 642  
 QueryInterface(), 584, 589, 666  
 rand(), 663

Read(), 548-549  
 ReadMail(), 680  
 RecalcLayout(), 474-475, 481  
 Rectangle(), 372  
 RedrawWindow(), 328-330  
 RegisterClipboardFormat(), 556-558  
 Release(), 583, 657, 672  
 ReleaseCapture(), 179, 613  
 ReleaseDC(), 667-668  
 ReleaseDispatch(), 601  
 RemoveAll(), 131  
 RemoveView(), 470  
 ReplaceSel(), 107  
 Requery(), 574  
 ResetContent(), 129  
 RestoreDisplayMode(), 672  
 RoundRect(), 372  
 SaveModified(), 257-258  
 Seek(), 553  
 SelectObject(), 352, 371  
 Serialize(), 259, 479, 533, 543  
 SetBitmap(), 231  
 SetBkColor(), 382  
 SetBkMode(), 383  
 SetCapture(), 179, 612  
 SetCheck(), 279  
 SetClipper(), 667  
 SetColor(), 164-165  
 SetCommand(), 195  
 SetCooperativeLevel(), 657  
 SetCursor(), 231  
 SetDateTimeSpan(), 167  
 SetEnhMetaFile(), 231  
 SetFirstDayOfWeek(), 164-165  
 SetFormat(), 157  
 SetIcon(), 231  
 SetImageList(), 238

SetLineSize(), 152  
 SetMapMode(), 337-339  
 SetModifiedFlag(), 258  
 SetMonthCalColor(), 160  
 SetMonthDelta(), 164-165  
 SetPageSize(), 152  
 SetPalette(), 667  
 SetPaneText(), 315  
 SetPixel(), 327, 332  
 SetPolyMode(), 375-376  
 SetPos(), 141-142  
 SetRange(), 165  
     *calendar, control,*  
     164-165  
     *indicator de evoluție,*  
     140  
     *selector de dată/oră,*  
     *control, 157*  
 SetRange32(), 140  
 SetScrollInfo(), 145  
 SetScrollPos(), 145  
 SetScrollRange(), 144  
 SetScrollSizes(), 417-418  
 SetSel(), 107  
 SetSelection(), 153  
 SetStep(), 141-142  
 SetSysString(), 591-592  
 SetTabStops(), 112  
 SetTextColor(), 382  
 SetTickFreq(), 152  
 SetTimer(), 672  
 SetToday(), 164-165  
 SetViewportExt(), 341  
 SetWindowExt(), 341  
 SetWindowLong(), 434  
 SetWindowPlacement(), 404  
 SetWindowPos(), 412  
 SetWindowText(), 78, 681  
 ShowControlBar(), 302  
 ShowWindow(), 76-79  
 SquareRoot(), 594  
 StartDoc(), 528  
 StartPage(), 529-530  
     static, 555  
 StepIt(), 140-142  
 Stop(), 658  
 strcmp(), 130  
 TabbedTextOut(), 379  
     text, afișare  
         *afișare simplă a*  
         *textului, 379*  
         *alinie, 380-382*  
         *culori de*  
         *afișarelfundal, 382*  
         *decupare la*  
         *dreptunghiuri,*  
         384-385  
         *text opac/transparent,*  
         383  
 TextOut(), 379  
     tip, 43  
 TrackPopupMenu(), 281-282  
 TrackRubberBand(), 182-184  
 TranslateColor(), 618  
     tratare, funcții  
         *adăugare, 276-279*  
         *bare de derulare,*  
         146-150  
         *butoane de pe bara cu*  
         *instrumente,*  
         299-300  
         *clic asupra*  
         *butoanelor, tratare,*  
         79-82  
         *comutatoare,*  
         *elemente de meniu,*  
         279  
         *create, 265-266*  
         *editare, 267-268*  
         *etichete, elementelor*  
         *de meniu, 280*  
         *subclase, 110*  
 Unlock(), 658  
 UnlockWindowUpdate(), 407-408

UpdateAllViews(), 495  
 UpdateData(), 87, 99,  
 207-208, 552  
 ValidateAge(), 213  
     variabile, 41  
     virtual, 508, 543  
         *clase abstracte, 583*  
         COM, 582  
 WaitForVerticalBlank(),  
 667-668  
 Width(), 359  
 Write(), 550  
**funcții constructor**  
     afișare, 203  
     CModeless, 214  
     coloane din  
     reprezentările de tip  
     listă, 434  
**fundal**  
     bare suport (Internet  
     Explorer), 321  
     culoarea textului, 382  
     pensule, 367  
     ștergere, 334

## G

**galeria de componente**  
 (controale ActiveX), 186  
     adăugarea controalelor,  
     188-189  
     inspectare, 186-187  
     testarea controalelor,  
     190  
**Game SDK, 655-656**  
**GDI (interfață pentru**  
**dispozitivele grafice).**  
     *Vedeți grafică*  
**GDI.DLL, 323**  
**generator de programe.**  
     *Vedeți AppWizard*  
**generice, proprietăți**  
 (controale ActiveX),  
 617  
**gestionare, clase (casete de**  
**dialog), 200**

gestionarea fișierelor,  
     aplicații (ferestre  
     multi-sectiune), 468-469  
 GetActiveDocument(),  
     funcție, 489  
 GetActiveView(), funcție,  
     474-475  
 GetAttachedSurface(),  
     funcție, 671  
 GetBackColor(), funcție,  
     188-189  
 GetBkColor(), funcție, 382  
 GetChannelRect(), funcție,  
     152  
 GetClientRect(), funcție,  
     178, 359, 420  
 GetComputerName(),  
     funcție, 98-99  
 GetCopies(), funcție, 521  
 GetCurrentDirectory(),  
     funcție, 124  
 GetCurrentPosition(),  
     funcție, 356-358, 397  
 GetCurSel(), funcție, 124  
 GetDatabase(), funcție, 571  
 GetDC(), funcție, 667-668,  
     672  
 GetDefaultConnect(),  
     funcție, 574  
 GetDefaultSQL(), funcție,  
     574  
 GetDesktopWindow(),  
     funcție, 327  
 GetDeviceCaps(), funcție,  
     342-344, 512  
 GetDlgCtrlID(), funcție,  
     154  
 GetDlgItem(), funcție, 72,  
     76, 106, 154  
 GetDocument(), funcție,  
     262-264, 489  
 GetDriverName(), funcție,  
     521  
 GetEditControl(), funcție,  
     448  
 GetFileName(), funcție,  
     554  
 GetFirstViewPosition(),  
     funcție, 474-475, 489  
 GetFromPage(), funcție,  
     521  
 GetHeadPosition(), funcție,  
     431  
 GetIDsOfNames(), funcție,  
     590  
 GetLBText(), funcție, 125  
 GetLine(), funcție, 112  
 GetLineCount(), funcție,  
     112  
 GetLineSize(), funcție, 152  
 GetMonitorFrequency(),  
     funcție, 667-668  
 GetMonthDelta(), funcție,  
     164-165  
 GetNextItem(), funcție, 438  
 GetNextView(), funcție,  
     474-475, 489  
 GetNumTicks(), funcție,  
     153  
 GetPageSize(), funcție, 152  
 GetParentFrame(), funcție,  
     489  
 GetPolyFillMode(), funcție,  
     375-376  
 GetPortName(), funcție,  
     521  
 GetPrinterDC(), funcție,  
     522  
 GetProcAddress(), funcție,  
     675-676, 680  
 GetRange(), funcție, 165  
     calendar, control,  
     164-165  
     indicator de evoluție,  
     control, 140  
     selector de dată/oră,  
     control, 157  
 GetSafeHwnd(), funcție,  
     412  
 GetScanLine(), funcție,  
     667-668  
 GetScrollInfo(), funcție,  
     145  
 GetScrollLimit(), funcție,  
     424  
 GetScrollPos(), funcție, 145  
 GetScrollPosition(),  
     funcție, 420  
 GetSel(), funcție, 144  
 GetSelection(), funcție, 106  
 GetStatus(), funcție, 153  
 GetSystemDirectory(),  
     funcție, 553  
 GetTextAlign(), funcție,  
     380  
 GetTextColor(), funcție,  
     382  
 GetTextMetrics(), funcție,  
     463  
 GetTime(), funcție, 159  
 GetToday(), funcție,  
     164-165  
 GetToPage(), funcție, 521  
 GetTypeInfo(), funcție, 590  
 GetTypeInfoCount(),  
     funcție, 590  
 GetUserName(), funcție,  
     98-99  
 GetVersionEx(), funcție,  
     98-99  
 GetWindowDC(), funcție,  
     327  
 GetWindowPlacement(),  
     funcție, 404  
 GetWindowRect(), funcție,  
     359  
 GetWindowsDirectory(),  
     funcție, 124  
 GetWindowText(), funcție,  
     106  
 glisor, control, 149  
     *adăugare în casete de*  
     *dialog, 150-151*  
     canal rectangular, 152  
     Fire, aplicație, 141-142  
     gradații, 152-153  
     inițializare, 152  
     mesaje de notificare,  
     153-154

poziție, stabilire, 152  
 proprietăți, 151  
 urmărirea deplasării, 153-154  
 valori de incrementare/decrementare, 152  
 variabile, 151-152  
**global, identificador unic. Veдеți GUID**  
**globale, funcții, 98-99**  
**GlobalMemoryStatus(), funcție, 98-99**  
**gradații, 153**  
**grafică**  
 afișarea textului, funcții, 379  
 casete combinate, 116  
 contexte dispozitiv, 323, 335, 48-49, 371-372  
 desenare (clasa CDC), 324  
 Direct3D, 673-674  
 DirectDraw  
   accesare, 667  
   agregare, 666  
   comutarea suprafețelor, 667-668  
   crearea obiectelor, 666  
   dublu buffer, 667-668, 671-672  
   funcții GDI, 667  
   HAL, 665  
   HEL, 665  
   liste pentru decupare, 667  
   ModeX, 665  
   nivel de cooperare, 667  
   obiecte, 664-667  
   palete, 667  
   suprafețe, 667  
 DirectPlay, 674  
 DLL, 323

indicatori către obiecte, 353  
 liste de imagini, 238-243  
   bitmap-uri, 240  
   controale arbore, 242  
   controale imagine (casete de dialog), 230-231  
   controale listă, 119-121, 132-136, 242  
   pictograme, 242  
 modificarea dinamică a culorilor, 667  
 penițe  
   cercuri, 360-362  
   contexte dispozitiv, 352  
   create, 349  
   culoare, 350  
   curbe, 362-369  
   desenare, 355-357  
   dreptunghiuri, 359-360  
   elipse, 360-362  
   grosime, 349  
   linii, 357-358  
   penițe de stoc, 351  
   penițe nule, 351  
   poligoane, 364-365  
   punct de coordonate, 358-360  
   ștergere, 353-354  
   tipuri, 349-350  
 pensule, 364-365  
   bitmap-uri, 367-369  
   cercuri, 374-375  
   contexte dispozitiv, 371-372  
   create, 365  
   culoare, 365  
   dreptunghiuri, 372-374  
   elipse, 374-375  
   fundal, 367

hașurare, 365  
 modele, 367-369  
 pensule cu culorile sistem, 369-371  
 pensule de stoc, 369-371  
 pensule nule, 370  
 poligoane, 375-377  
 sectoare, 374-375  
 suprafețe de coardă, 374-375  
 ștergere, 372

perioada inertă dintre cadre, 672  
 tipărire, 504, 507-508, 515, 524  
 viteză, 655-656  
**Graphics, casetă cu instrumente, 223-224**  
**grilă (controale listă), 119-121, 132-136**  
**Group Box Properties, casetă de dialog, 83**  
**Group, proprietate, 83-88**  
**grupare**  
   butoane de opțiune, 83-85  
   casete de grupare, 68, 83  
   controale din casete de dialog, 68-69  
**GUID (identificador global unic), 584**  
   CLSID, 584  
   create, 584-586  
   IID, 584  
**guidgen.exe, 584-586**  
**GWL\_STYLE, indicator, 433**  
**g\_pDlgModeless, pointer, 220**

## H

.h, extensie de fișier, 50  
**HAL (nivel de abstrac-tizare hardware), 665**

**hardware, limitări asupra culorilor, 350**  
**hașurare, 365**  
**hărți de mesaje, 75-76**  
**hdc, membru, 334**  
**Height(), funcție, 359**  
**HEL (nivel de emulare hardware), 665**  
**HORZRES, indicator, 346**  
**HTML (Hypertext Markup Language), 454**

## I

**IAutoserver, clasă, 599**  
**IClassFactory (interfață COM), 587**  
 .ico, extensie de fișier, 50  
 Icon Properties, casetă de dialog, 226-227  
**IDC\_STATIC, identificador, 96**  
**IDD\_ABOUTBOX, machetă, 56-57**  
**IDD\_DIALOGBAR, machetă, 305**  
**IDD\_MINUTE\_DIALOG, resursă, 27**  
**IDD\_MOUSEMSG\_DIALOG, aplicație, 169**  
**IDD\_PERSON\_DIALOG, machetă, 56-57**  
**IDE (mediu integrat de dezvoltare), 21**  
 identificatori (controale etichetă statică), 96  
**IDL (limbaj de definiție a interfețelor), 594, 627**  
**IDR\_MDICOITYPE, identificador, 482**  
**ID\_INDICATOR\_MOUSE, identificador de șir, 313**  
 ierarhii, 116  
**IID (identificador de interfață), 584**  
 Images, casetă de dialog, 233

**imagine, control (casete de dialog)**

  adăugare, 232  
   adăugarea variabilelor, 233, 235  
   încărcare la momentul execuției, 231  
   tipuri de imagini, 230-231

**imagini, editor, 223**

**imagini, liste, 238-243**

  bitmap-uri, 240  
   controale arbore, 242  
   controale imagine (casete de dialog), 230-231  
   controale listă, 119-121, 132-136, 242  
   pictograme, 242

**imagini. Veдеți grafică**

**imediată, depanare, 638**

**imediată, editare, 447**

**implementare**

  clase, 536-539  
   controale ActiveX, 609  
**IMPLEMENT\_DYNCRE**  
   ATE, macrodefiniție, 254-255, 485

**implicite, butoane (casete de dialog), 63**

**implicită, procedura de fereastră, 111**

**Import Resource, casetă de dialog, 229**

**importul bitmap-urilor, 229-230**

**incrementarea variabilelor, 42-43**

**indicator de evoluție, control, 137**

  adăugare în casete de dialog, 137-139

  Fire, aplicație, 141-142

  interval, 140

  manipulare, 140

  poziție, 141-142

  stiluri, 138

  valoare de increment, 142

  variabile, 139-140

**indicatori**

  bară de controale

    ancorare, 295

    bară cu instrumente standard, 294-295

  CDC, clasă, 324

  CRectTracker, indicatori de stil, 182

  dimensionare, eveniment, 409

  DRIVERVERSION, 343

  ETO\_CLIPPED, 384-385

  fErase, 334

  GetNextItem(), funcție, 438, 446

  GWL\_STYLE, 433

  HORZRES, 346

  indicatori de meniu

    AppendMenu(),

      funcție, 286-287

    InsertMenu(), funcție, 286-287

  LVNI\_SELECTED, 438

  măști de biți (evenimente generate de butoanele mouse-ului), 171-172

  mod de deschidere (CFile), 548

  obiecte, 353

  OLECMDID\_SETPROG  
   RESSMAX, 456

  OPAQUE, 383

  PARAFORMAT, structură, 451

  reprezentare de tip arbore, stiluri, 442

  SetTextAlign(), funcție, 380

  SetWindowPos(), funcție, 412

  SRCINVERT, 336

stiluri  
     *bare suport*, 319-321  
     *bară de stare*, 316  
 TECHNOLOGY, 343  
 TRANSPARENT, 383  
 validări DDV specifice, 212  
 valori  
     *alinie*, 282  
     *bară cu instrumente*,  
       *margin* de  
       *ancorare*, 296  
     *bară de dialog*,  
       *alinie*, 306  
     *CFileStatus*, *atribute*,  
       554  
     *Seek()*, *funcție*, 553  
 indicatori (*bară de stare*),  
 310  
     poziția mouse-ului, 312  
     proprii, 312, 313-314  
 indicatori, vector, 310  
 InflateRect(), *funcție*, 362  
 infrastructuri  
     ActiveX, 605-609  
     document, 504  
 Initialize(), *funcție*, 667,  
 671  
 InitialUpdateFrame(),  
*funcție*, 258, 488  
 InitInstance(), *funcție*, 253,  
 484, 595-596  
 inițializare  
     câmpuri, 102-104  
     CMenu, obiect, 285-286  
     MDI, aplicații, 483-486  
     tipărire, 523, 529-530  
     variabile, 42  
 inserare  
     bare cu instrumente,  
       resurse, 300  
     bitmap-uri, 227-228  
     pictograme, 226-227  
 Insert ActiveX Control,  
 casetă de dialog, 191

Insert Object, casetă de  
 dialog, 607  
 Insert Resource, casetă de  
 dialog, 45, 201, 226-227  
 Insert, meniu, comanda  
 Resource, 226-228  
 InsertColumn(), *funcție*,  
 133  
 InsertItem(), *funcție*, 125,  
 243, 438, 443  
 InsertMenu(), *funcție*,  
 286-287  
 inspectarea codului  
 (Source Browser),  
 635-637  
 inspectarea fișierelor, 41  
 instalarea extensiilor  
 DirectX, 674  
 instantanee, 569-570  
 interfață document  
 multiplu. *Vedeți* MDI  
 interfață document  
 singular. *Vedeți* SDI  
 interfață pentru controlul  
 multimedia. *Vedeți* MCI  
 interfață pentru  
 programarea aplicațiilor  
 de mesagerie. *Vedeți*  
 MAPI  
 interfață, identificator  
 (IID), 584  
 interfețe. *Vedeți* și API;  
 reprezentări; ferestre  
     ActiveX, controale, 187,  
       617-627  
     asocierea codului, 30,  
       32  
     buton, controale, 26-27,  
       28-29  
     COM, 581  
     *convenții pentru*  
       *denumire*, 582  
     *IClassFactory*, 587  
     *IUnknown*, 583  
     *versiuni*, 588-589  
     Direct3D, 673

GUID  
     CLSID, 584  
     creare, 584-586  
     IID, 584  
 interfețe duale, 590  
 MCI, fereastră, 689-692  
 procesoare de text,  
 reprezentare de tip  
 editare formatată, 449  
 SDI, aplicații, 251-253  
 Windows, 26  
 intermediere  
     biblioteci DLL de  
       intermediere, 588  
     suport pentru  
       automatizarea OLE,  
       590-529  
 Internet Explorer  
     bară cu instrumente, 294  
     bare suport  
       *ancorare*, 319  
       *AppWizard*, 319-321  
       *bitmap de fundal*,  
       319-321  
       *controale încapsulate*,  
       320  
       *stiluri*, 319-321  
       *titlu*, 321  
     browser HTML, tip de  
       reprezentare, 454  
 interogări (SQL)  
     actualizarea valorilor,  
       575  
     aplicații de baze de date,  
       572-575  
     schimb de câmpuri, 576  
 intervale de valori,  
 controale orientate spre,  
 137, 165  
 intra-proces, server  
 (contextul procesului  
 server), 586  
 intrare  
     bare de controale  
       *focus*, 300  
       indicatori, 294-296

butoanele mouse-ului.  
*Vedeți* mouse  
 casete de editare  
     *adăugare*, 100-102  
     *adăugare de*  
       *variabile*, 102-103  
     *determinarea textului*,  
       102-104  
     *distincție minuscule/*  
       *majuscule*, 100  
     *inițializarea*  
       *câmpurilor*, 102-104  
     *mesaje de notificare*,  
       104-105, 107  
     *moștenirea claselor*,  
       107-112  
     *multi-linie*, 112  
     *numere*, 100  
     *parole*, 100  
     *taste mnemonice*, 102  
     *testare*, 102  
     *tratarea mesajelor*,  
       104-105  
     DirectInput, 674  
 introducerea datelor.  
*Vedeți* introducere  
 Invalidate(), *funcție*, 175,  
 421-422  
 InvalidateControl(),  
*funcție*, 612  
 InvalidateRect(), *funcție*,  
 421-422  
 InvalidateRgn(), *funcție*,  
 421-422  
 Invoke(), *funcție*, 590, 600  
 InvokeHelper(), *funcție*,  
 188-189, 599  
 IsEmpty(), *funcție*, 104  
 IsIconic(), *funcție*, 176, 331  
 IsKindOf(), *funcție*, 474  
 IsPrinting(), *funcție*, 511  
 IsWindowEnabled(),  
*funcție*, 80  
 IsWindowVisible(), *funcție*,  
 78

iteratori, 393  
 IUnknown, interfață  
 (COM), 583

## I

încadrare (*încadrare*  
*rectangulară*), 181-184  
 încadrare rectangulară,  
 181-184  
 încapsulare  
     contexte dispozitiv, 324  
     CPaintDC, 330  
 încapsulare, clase  
     CFile, 545-546  
     CFont, 386  
 încapsularea datelor, 248  
     controale, bare suport,  
       320  
     documente, identificatori,  
       603  
     OLE (legarea și  
       încapsularea  
       obiectelor), 602-604  
       CHtmlView\_ExecWB(),  
       *indicatori de comandă*  
       *pentru funcție*, 455  
       Clipboard, 556  
       *depanarea obiectelor*,  
       639  
       *editare formatată, tip*  
       *de reprezentare*, 453  
       *fișiere compuse*, 589  
       *interfețe duale*, 590  
       *moniker*, 589  
     parametri în linie de  
       comandă, 595-596  
 început/sfârșit, pagini  
 (tipărire), 515-518  
 închidere  
     casete de dialog modale,  
       205  
     casete de dialog  
       nemodale, 220-221  
     proiecte, 33  
 înclinare, 388

înregistrare  
     controale ActiveX,  
       626-628  
     subclasarea controalelor,  
       627  
     tipuri de document,  
       545-546  
 înregistrare, tip de  
 reprezentare  
     controale de editare,  
       578-579  
     creare, 576-577  
     editare, 577-578  
 întrerupere, proceduri  
 (MCI), 685  
 întreținere, casetă de  
 dialog, 206

## J

jocuri, DirectPlay, 674  
 jurnalizarea evenimentelor  
 generate de controalele  
 ActiveX, 630

## L

la distanță, depanare, 638  
 la distanță, server  
 (contextul procesului  
 server), 586  
 lansarea Visual C++, 21  
 Layout, meniu  
     comenzi  
       *Tab Order*, 84-85  
       *Test*, 84-85, 102, 614  
     Developer Studio, 60  
 legare, 24-25  
 legarea și încapsularea  
 obiectelor. *Vedeți* OLE  
     documente, 603  
     OLE (legarea și  
       încapsularea obiectelor)  
       CHtmlView\_ExecWB(  
       ), *funcție*, *indicatori*  
       *de comandă*, 455  
       Clipboard, 556

depanarea obiectelor, 639

editare formatată, tip de reprezentare, 453

fișiere compuse, 589

interfețe duale, 590

moniker, 589

licențierea controalelor

ActiveX, 605-606, 628

limbaj pentru controlul datelor (DCL), 565-566

limbaj pentru definirea datelor (DDL), 565-566

limbaj pentru manipularea datelor (DML), 565-566

limbaj structurat de interogare. *Vedeți SQL*

LimitText(), funcție, 107

LineTo(), funcție, 357-358

linie de comandă, parametri

analiză, 255-256

automatizare, 595-596

încapsulare, 595-596

linie nouă, caracter (\n), 94

linie, increment de derulare, 419

linii de ghidare, 66-67, 305

linii, desenare, 357-358

lipire, 560-562

listă, tip de control, 238-240, 243

arbore, control

creare, 116-117

noduri, 116

populare, 125-128

variabile, 117

casetă cu listă, control

adăugare în caseta de dialog About, 344

arbore, subclase, 117

creare, 118-119

mesaje de notificare, 130-131

populare, 128-130

variabile, 119

casete combinate

creare, 115-118

derulare, 114-115

determinarea conținutului, 124-125

imagini, 116

mesaje de notificare, 124-125

populare, 121-124

sortare, 115-116

variabile, 115-116

creare, 119-121

listă de imagini, tipuri, 242

listă, control

creare, 119-121

populare, 132-136

prezentare, 115

reprezentare, 120

selectarea liniilor, 432

listă, tip de reprezentare

coloane și antete, 434, 437-438

creare, 427

determinarea elementelor selectate, 438-440

inserarea elementelor, 427-430

modificarea stilurilor, 432-434

liste

determinarea butoanelor de opțiune, 86-87

fișiere, 127

fișierele utilizate recent, 545-546

listinguri

ActiveX, controale

acțiunile utilizatorului, 612-613

desenare, 609-612

OnDraw(), funcție, 626-628

permanența proprietăților, 624-625

proprietăți specifice, 621-622

tratarea evenimentelor, 612-613

arbore, control, 125-128

audio, buffere

DirectSound, 658-659, 662-664

automatizare, servere

limbaj de definiție a obiectelor, 593-596

bară de dialog, funcții, 307-308

bară de stare

poziția mouse-ului, 318

secțiuni indicatoare, 317

bare de derulare, tratarea mesajelor și actualizarea poziției, 149

Blob, date, clasa Object

definiție, 535-536

implementare, 537-538

casete combinate

determinarea conținutului, 124

populare, 122

casete cu listă

mesaje de notificare, 130-131

populare, 129-130

casete de dialog

OnOK(), funcție implicită, 48

titlu, 80-82

casete de editare

determinarea textului, 102-103

mesaje de notificare, 105-106

OnChar(), funcție de tratare, 110

casete de validare, 89-92

control de editare

citirea conținutului unui fișier, 550

scrierea datelor într-un fișier de pe disc, 551

CSingleDocTemplate, clasă, funcția

InitInstance(), 253-254

CUsingDBSet, clasă, 572-573

desenare

cercuri (funcția Ellipse()), 360-362

CPoint, clasă, 359

CRect, clasă, 359

curbe (funcția PolyBezier()), 362-369

DirectDraw, dublu buffer, 667-668, 671-672

dreptunghiuri, 372-373

obiecte blob, 541

poligoane, 375-377

Polyline(), funcție, 364-365

sectoare de coardă, 375

DoFieldExchange(), funcție, 576-577

elemente de meniu

comutator, marcare de validare, 279

etichetă statică, control, 99

ferestre multi-sectiune

ferestre multi-sectiune dinamice, 463

ferestre multi-sectiune statice, 465-467

fundal, funcția

OnEraseBkGnd(), 367

imagini

liste, 240-242

resurse, 234-235

incrementarea variabilei contor, 42-43

inițializarea variabilelor membru, 42

listă, control

inițializarea coloanelor, 132

populare, 133-135

MAPI, suport în cadrul infrastructurii, 677-682

MCI, mesaje de notificare, 685-688

MDI, aplicații

accesarea datelor, 492-493

actualizarea reprezentărilor, 494-495

crearea reprezentărilor, 486-488

inițializarea machetelor de documente, 499-500

meniuri contextuale, 288

mesaje, pe baza unei variabile, 43

mulțimi de înregistrări, parcurgere, 574

OnContextMenu(), funcție, 283

OnLButtonDown(), funcție, 170-171

OnMouseMove(), funcție

poziția cursorului, 175-176

urmărirea cursorului, 177-178

OpenEx(), funcție, 571

penițe

CreateObject(), funcție, 353-354

culori și stiluri, 351

DeleteObject(), funcție, 353-354

LineTo(), funcție, 357-358

MoveTo(), funcție, 356-357

selectare, 352-353

pensule

bitmap-uri, 369

pensule cu culorile sistem, 371

selectare, 371

ProcessShellCommand, funcție, 256-257

reprezentări

controlul activării, 471-475

imagini, accesarea datelor documentului, 263-264

selector de dată/oră, control, mesaje de notificare, 161-162

ShowWindow(), funcție

activarea/dezactivarea a butoanelor, 78-79

afișarea/ascunderea butoanelor, 76-78

TRACE, macrodefiniție, exemplu, 640-641

Lists, casetă de dialog, 114-115

LoadBitmap(), funcție, 234-235, 369

LoadFrame(), funcție, 258, 485, 488

LoadLibrary(), funcție, 675-676

LoadMenu(), funcție, 285

LoadStandardIcon(), funcție, 171-172

LoadToolBar(), funcție, 294

Local, server (contextul procesului server), 586



Location, casetă, 23  
 Lock(), funcție, 658  
 LockWindowUpdate(), funcție, 407-408  
 logice, unități, 337-339  
 LParam, parametru, 395  
 lpszFacename parametru, 390  
 LVNI\_SELECTED, indicatori, 438  
 LVS\_STYLE, stil, 432

## M

### machete

bară de dialog, resursă, 304  
 casete de dialog  
   *casete de dialog nemodale*, resurse, 213  
   *denumire*, 58-60  
   *dimensionare*, 62-63  
   *proiectare*, 54-58  
   *proprietăți*, 58  
   *stiluri*, 59-60  
 casete de dialog, controale  
   *aliniere*, 66  
   *aspect*, 60, 62-64  
   *dimensionare*, 64  
   *editare*, 65  
   *grupare*, 68-69  
   *linii de ghidare*, 66-67  
   *ordine de selectare*, 69-70  
   *taste de acces*, 70-71  
 clase modale, 205  
 IDD\_ABOUTBOX, 56-57  
 IDD\_DIALOGBAR, 305  
 IDD\_PERSON\_DIALOG, 56-57  
 machete de document  
   *adăugarea opțiunilor de meniu*, 265-266

### machete de document

MDI, aplicații, 483-486  
   *clase*, 496-497  
   *creare*, 495-500, 502  
   *inițializare*, 499-500  
   *meniuri*, 497-498  
   *șiruri*, 498-499  
 SDI, aplicații, 253  
   *adăugarea opțiunilor de meniu*, 265-266  
   *Document/Reprezentare*, funcțiile  
     *infrastructurii*, 255-260

### macrodefiniții

ASSERT, 642-644  
 ASSERT\_NULL\_OR\_POINTER, 643  
 ASSERT\_POINTER, 643  
 BEGIN\_MESSAGE\_MAP, 76  
 DECLARE\_DYNAMIC\_CREATE, 254-255, 485  
 DISP\_FUNCTION, 597  
 IMPLEMENT\_DYNAMIC\_CREATE, 254-255, 485  
 RFX, 576-577

*aplicații MDI*, 483-486, 495-502  
*aplicații SDI*, 253-260, 265-266  
*Document/Reprezentare*, funcțiile  
   *infrastructurii*, 255-260  
   *serializarea șirurilor*, 532-533

machete de reprezentări de tip înregistrare  
   *controale de editare*, 578-579  
   *creare*, 576-577  
   *editare*, 577-578

RUNTIME\_CLASS, 254-255, 484  
 TRACE, 639-642  
 VERIFY, 642-644  
 Mail and Fax, panoul de control, 675-676  
 MakeReverse(), funcție, 104  
 mapare  
   moduri, 337-340  
     *caracteristicile dispozitivelor*, 342-347  
     *scalare liberă*, 340-342  
   proprietățile controalelor  
     ActiveX, 622-624  
 mapate, variabile membru (DDV), 210  
 MAPI (interfață pentru programarea aplicațiilor de mesagerie)  
   COM, 581-582  
   funcții, 674-677  
   MAPI extins, 675  
   SDK, 675  
   suport în cadrul  
     *infrastructurii*, 677-681  
 MAPIFreeBuffer(), funcție, 681  
 MapMode, clasă reprezentare, 339  
 marcaje de validare  
   afișare/înălțare (elemente de meniu), 279-280  
   afișarea alături de elemente de meniu, 274-275  
 math.h, fișier de antet (subclasarea controalelor), 611  
 măști de biți, 451  
 MCI (interfață pentru controlul multimedia), 681, 190  
   comenzi, 682-683

dispozitive, 682-683  
 fereastră MCI, 689-692  
 mesaje de notificare, 685-688  
   proprietăți, 192  
 mciGetErrorString(), funcție, 688  
 mciGetYieldProc(), funcție, 685  
 mciSendCommand(), funcție, 682  
 mciSendString(), funcție, 685  
 mciSetYieldProc(), funcție, 690  
 MCIWndChangeStyles(), funcție, 691  
 MCIWndGetMode(), funcție, 690  
 MCIWndGetStyles(), funcție, 675  
 MDBView.exe, program  
 MDI (interfață document multiplu), aplicații  
   bare cu instrumente, 482  
   bare de stare, 482  
   clase, 480-482  
   creare, 478  
   date  
     *accesare*, 492-493  
     *modificare*, 493-496  
     *stocare*, 490-492  
 documente  
   *accesare*, 488  
   *creare*, 485-489  
   *meniuri*, 482  
   *pictograme*, 482  
   *încapsularea datelor*, 479  
   inițializare, 483-486  
   machete de document, 483-486  
     *clase*, 496-497  
     *creare*, 495-500, 502  
     *inițializare*, 499-500

*meniuri*, 497-498  
   *șiruri*, 498-499  
 meniuri, 482  
 reprezentări, 481-483  
   *actualizare*, 493-496  
   *reprezentare activă*, 481  
 resurse, 481-483  
 serializare, 532  
 tabele de șiruri, 482  
 tipărire, 504  
 variabile membru, 490-492  
 MDIGetActive(), funcție, 489  
 mediu integrat de dezvoltare (IDE), 21  
 membru, funcții  
   casete de dialog  
     *nemodale*, 216  
   ClassView, 80  
   CListCtrl, clasă, 132  
   CListVDoc, clasă, 428  
   CPrintDialog, clasă, 521  
   creare, 471  
   CStatic, clasă, 231-232  
   funcții membru publice, 481

### membru, variabile

casete de dialog  
   *nemodale*, 217  
 ClassView, 80  
 DDV, 210  
 dialog, clase, 206-207  
 dispecer, clase, 193  
 document, clase, 261-262  
 MDI, aplicații, 490-492  
 OnMouseMove(), funcție, 176  
 protejate, 261-262, 492

### memorie

bit blitting, 336  
 buffer pentru semnal de undă, 655-656

memorie virtuală, 99  
 modificarea dinamică a culorilor, 667  
 obiecte COM, 584  
 memorie, context  
   dispozitiv, 335-336  
 Memory, fereastră (depanator), 649  
 meniuri  
   CMenu, obiect, 285-286  
   comenzi  
     *activare/dezactivare*, 278  
     *identificatori*, 273  
     *tratare*, 276-280  
   editare formatată, tip de reprezentare, 451  
   elemente  
     *adăugare dinamică*, 286-288  
     *editare dinamică*, 289-290  
     *elemente de meniu*, 271-272  
     *etichete*, 280  
     *marcaje de validare*, 274-275, 279-280  
     *proprietăți*, 273  
     *separatori*, 274  
     *submeniuri*, 274  
     *ștergerea dinamică*, 290  
     *taste de acces*, 274-275  
     *titluri*, 271  
   identificatori, 286-287  
   indicatori  
     AppendMenu(), funcție, 286-287  
     InsertMenu(), funcție, 286-287  
 MDI, aplicații, 482  
 meniuri de context  
   *adăugare*, 280  
   *afișare*, 281-283



- funcții de tratare pentru meniurile de context, 281-282  
tratarea comenzilor din meniurile de context, 284  
resurse, 46, 270-271
- Menu Item Properties**, casetă de dialog, 265, 272, 494
- mesaje (casete de dialog), 200
- mesaje de notificare  
casete combinate, 124-125  
casete de editare, 104-105, 107  
controale casetă cu listă, 130-131  
MCI, 685-688
- mesaje pentru utilizator, casete de dialog, 200
- mesaje. *Vedeți* evenimente
- Message Options**, casetă de dialog, 650-651
- MessageBox()**, funcție, 87
- metafișiere, 231
- metoda generatorului de clase, 254-255
- metode. *Vedeți* funcții
- MFC** (biblioteca de clase de bază Microsoft), 7. *Vedeți* și clase
- AppWizard, 23
- CBrush, clasă, 365
- CEdit, clasă, 106
- context dispozitiv, 324
- CPreviewDC, clasă, 518
- CreateFont(), funcție, 387
- CScrollView, clasă, 415
- CStringList, clasă, 427-428
- CView, clasă, 508-509
- pictograme, 225
- MFC Assert**, casetă de dialog, 474-475
- MFC ClassWizard**, casetă de dialog, 240
- MFC Tracer**, comandă (meniul Tools), 575
- MFC Tracer**, instrument, 653
- m\_hAttribDC, indicator, 324
- m\_hDC, indicator, 324
- Microsoft Plus!, 407-408
- Microsoft Sound .WAV, fișier, 195
- MIDL** (limbaj de definiție a interfețelor de la Microsoft), 594, 627
- MINMAXINFO**, structură, 414
- minuscule/majuscule, distincție, 100
- Minute.exe, 24-25
- MkTypLib**, 594, 627
- MM\_ANISOTROPIC**, mod de mapare, 339-341
- MM\_ISOTROPIC**, mod de mapare, 339, 342
- MM\_LOMETRIC**, mod de mapare, 337-338
- MM\_TEXT**, mod de mapare, 337-339
- modale, casete de dialog, 200
- afișare, 204-205
- casete de dialog modale față de sistem, 220
- închidere, 205
- modele**  
interferență moiré, 327
- pensule, 367-369
- modelul componentelor obiectuale. Vedeți** COM
- ModeX**, 665
- ModifyMenu()**, funcție, 289-290
- moiré, interferență, 327
- momentul execuției, operații**  
butoane de comandă, 82  
activare/dezactivare, 78-79  
afișare/ascundere, 76-78
- etichetă statică, control, 95-100
- ferestre, creare, 461-462
- funcții de tratare, efectuare de clic, 79-82
- resurse imagine, încărcare, 231-236
- moniker**, 589  
documente compuse, 603  
tabela cu obiectele aflate în rulare, 592
- monitoare, perioada inertă dintre cadre**, 672
- mouse**  
bară de stare, indicatori, 312  
captura acțiunilor, 179-180  
capturare, 179  
deplasare, eveniment  
parametri, 175  
poziția cursorului, listing, 175-176  
urmărire, 175  
urmărirea cursorului, listing, 177-178  
variabile membru, 176
- evenimente generate de butoane  
dublu clic, eveniment, 174  
interceptare, 169  
tratarea mesajelor, 169-173
- puncte senzitive, 180
- redimensionarea ferestrei, eveniment, 406
- teste de clic, 180-181
- MoveTo()**, funcții, 355-357

- multi-linie, casete de editare**, 112
- multi-linie, controale de editare derulabile**, 409-410
- multi-secțiune, ferestre**  
ferestre multi-secțiune dinamice  
creare, 458-459  
inițializare, 461-463
- ferestre multi-secțiune statice  
creare, 464  
crearea aplicațiilor de gestionare a fișierelor, 468-469  
inițializare, 464-467
- prezentare, 458
- Multiple Selection Properties**, casetă de dialog, 73
- mulțimi de înregistrări**  
actualizarea valorilor, 575  
câmpuri  
controale de editare, 578-579  
schimburi de date, 576
- instantanee, 569-570
- mulțimi dinamice, 569-570
- parcursare, 574
- muzică. Vedeți** audio

## N

- Navigate2()**, funcție, 454
- .ncb**, extensie de fișier, 50
- nCharSet**, parametru, 389
- nClipPrecision**, parametru, 389
- nemodale, casete de dialog**, 200, 213  
creare/distrugere, 213-216  
încheiere, 218
- mesaj de închidere, 220  
opțiune de închidere, 221
- nEscapement**, parametru, 388
- New Class**, casetă de dialog, 108, 464-465
- New Document**, casetă de dialog, 485
- New Icon Image**, casetă de dialog, 226
- New Project Information**, casetă de dialog, 23, 248, 479
- New Project**, casetă de dialog, 54
- New Windows Message and Event Handlers**, casetă de dialog, 170
- NEWTEXTMETRICS**, structură, 397
- nHeight**, parametru, 387
- nivele de cooperare  
DirectDraw, 667  
DirectSound, 657, 663
- noduri (controale arbore)**, 116
- nOrientation**, parametru, 388
- North**, opțiune de meniu, 284
- nOutPrecision**, parametru, 389
- nouă, resursă de tip machetă de dialog, 201
- nPitchAndFamily**, parametru, 389
- nQuality**, parametru, 389
- nul, obiect  
penițe, 351  
pensule, 370  
șir cu terminator, 550-551
- NULL**, caracter, 550, 351
- numerice, casete de editare, 100
- nWidth**, parametru, 388

## O

- obiecte**  
CDialogBar, 304-305  
CFileException, 547  
CImageList, 238-242  
COM  
creare, 586-588  
eliberarea memoriei, 584
- CToolBar, 293, 299-300
- DirectDraw, 664-667
- directe (Direct3D), 673
- elaborate (Direct3D), 673
- indicatori, 353
- înregistrarea clienților de automatizare, 600
- MDI, aplicații, 486-489
- obiecte meniu, 285-286
- serializabile, 533-536, 539
- serializare, 543-545
- obiecte de clasă (COM)**, 586-588
- obiecte, limbaje de definiție**, 593-596
- obiectuale, baze de date**, 564
- ODBC (conectivitate deschisă pentru bazele de date)**  
aplicații  
distribuire, 564-565  
suport, 569
- clase DAO, 571
- configurarea surselor de date, 566-568
- SQL, 565-566
- ODBC Data Source Administrator**, casetă de dialog, 566
- ODBC Microsoft Access Setup**, casetă de dialog, 567
- OffsetPos()**, funcție, 141-142

**OLE (legarea și încapsularea obiectelor)**

CHtmlView\_ExecWB(), indicatori de comandă pentru funcție, 455  
Clipboard, 556  
depanarea obiectelor, 639  
editare formatată, tip de reprezentare, 453  
fișiere compuse, 589  
interfețe duale, 590  
moniker, 589

**OLE DB SDK, 655****OLE, automatizare, 589**

clienți de automatizare  
  *clasă driver de dispecer*, 599-601  
  *creare*, 599-601  
interfață dispecer, 589-590  
servele de automatizare  
  *adăugarea metodelor*, 597-599  
  *COleObjectFactory*, 595-596  
  *containere*, 602-604  
  *creare*, 591-597  
  *documente active*, 602  
  *încapsulare*, 602-604  
  *limbaj de definiție a obiectelor*, 593-596  
  *nume*, 591-592  
  *registru*, 592  
  *rulare în fundal*, 592  
  *tabela cu obiectele aflate în rulare*, 592  
  *VB Scripting*, 539  
  *WordPad*, 603  
suport pentru apeluri intermediare, 590-529

**OLE/COM Object Viewer, instrument, 653****OLECMDID\_SETPROGR**

ESSMAX, indicator, 456  
OLE\_COLOR, tip de date, 618

OnActivate(), funcție, 481

OnActiveView(), funcție, 474-475

OnBeforeNavigate2(), funcție, 455

OnBeginPrinting(), funcție, 523

OnChangedViewList(), funcție, 470

OnChangeEditSecond(), funcție, 105

OnChar(), funcție, 110

OnChoosefont(), funcție, 397-398

OnClickCalendar(), funcție, 196

OnClose(), funcție, 220

OnCloseDocument(), funcție, 259, 470

OnContextMenu(), funcție, 283

OnCreate(), funcție, 293, 461

OnCreateClient(), funcție, 461

OnDestroy(), funcție, 327

OnDoVerb(), funcție, 603

OnDownloadBegin(), funcție, 456

OnDownloadComplete(), funcție, 456

OnDraw(), funcție, 263-264, 339, 355-357, 504-508

  alinieră textului, 380-381

  comparație cu OnPrint(), 508-509

OnDrawit(), funcție, 329, 332

OnDrawSplitter(), funcție, 464-465

OnEditCopy(), funcție, 558

OnEnableDisable(), funcție, 78

OnEndlabeledit(), funcție, 447

OnEndPrinting(), funcție, 523

OnEraseBkgnd(), funcție, 366, 372

OnFileNew(), funcție, 257-258

OnFileReceive(), funcție, 680

OnGetMinMaxInfo(), funcție, 413-414

OnGetText(), funcție, 103

OnHScroll(), funcție, 424

OnInitDialog(), funcție, 132

OnInitDialog(), funcție, 132, 164-165, 194, 240, 600

OnInitialUpdate(), funcție, 258, 416-417, 463, 489, 574

  inserarea elementelor în cadrul unei reprezentări de tip arbore, 443

  listă, tip de reprezentare, 430

OnKeyDown(), funcție, 663

OnLButtonDown(), funcție, 612

  capturarea mouse-ului, 179

  indicatori, 171

  listing, 170-171

OnMouseMove(), funcție, 612

  parametri, 175

  poziția cursorului, listing, 175-176

  urmărirea cursorului, listing, 177-178

  variabile membru, 176

OnNewDocument(), funcție, 259

OnOk(), funcție, 86

OnOpenDocument(), funcție, 259

OnPaint(), funcție, 176

OnPreparePrinting(), funcție, 515

OnPressMe(), funcție, 32

OnPrint(), funcție, 504, 508-510

OnProgressChange(), funcție, 456

OnSaveDocument(), funcție, 259

OnScroll(), funcție, 421-422

OnSelchangeSubDirs(), funcție, 130

OnSelMainDir(), funcție, 124

OnSetFocusEditShow(), funcție, 106

OnShowArt(), funcție, 471

OnShowEdit(), funcție, 471

OnShowHide(), funcție, 74

OnSize(), funcție, 411

OnSizing(), funcție, 406

OnStartModeless(), funcție, 215

OnStopModeless(), funcție, 216

OnTimer(), funcție, 672

OnUpdate(), funcție, 495

OnUpdateMousePos(), funcție, 313

OnUpdateShowArt(), funcție, 474

OnUpdateShowEdit(), funcție, 474

OPAQUE, indicator, 383

Open As Resources, comandă (meniul File), 229

Open(), funcție, 547, 574

OpenDocumentFile(), funcție, 257-258, 488

OpenEx(), funcție, 570-572

operații

  butoane de comandă, 82

*activare/dezactivare*, 78-79

*afișare/ascundere*, 76-78

  etichetă statică, control, 95-100

  ferestre, creare, 461-462

  funcții de tratare, efectuare de clic, 79-82

  întârzieri la execuția controlată, 646

  resurse imagine, încărcare, 231-236

  .opt, extensie de fișier, 50

  optimizarea tipăririi, 525

  Optimizations, 634

  ordine de selectare (casete de dialog), 69-70

  orientare, 388, 515

  orientată pe obiect, proiectare, 428

  Outlook (Word, server de automatizare), 589

  Output, fereastră, 36

**P****pagini**

  increment de derulare, 419

  tipărire, 515

PAINTSTRUCT, structură, 334

palette, 667

panoul de control (Mail and Fax), 675-676

PARAFORMAT, structură, 451

**parametri**

  AppendMenu(), funcție, 286

  DockControlBar(), funcție, 296

  linie de comandă

*analiză*, 255-256

*automatizare*, 595-596

  ModifyMenu(), funcție, 289-290

OnMouseMove(), funcție, 175

Read(), funcție, 548-549

szBuffer, 124

TrackPopupMenu(), funcție, 282

**parole, 100**

ParseCommandLine(), funcție, 255-256, 595-596

ParseParam(), funcție, 255

părinte/copil, relație, 116

pDX, pointer, 208

penițe

  contexte dispozitiv, 352-353

  creare, 349

  culoare, 350

  desenare, 355

*cercuri*, 360-362

*contexte dispozitiv*, 355

*curbe*, 362-369

*dreptunghiuri*, 359-360

*elipse*, 360-362

*linii*, 357-358

*poligoane*, 364-365

*puncte de coordonate*, 358-360

  figuri umplute, 349

  grosime, 349

  penițe de stoc, 351

  penițe nule, 351

  selectare, 352-353

  ștergere, 353-354

  tipuri, 349-350

**pensule, 364-365**

  bitmap-uri, 367-369, 372-374

  contexte dispozitiv, 371-372

  creare, 365

  culoare, 365

  desenare

*cercuri*, 374-375

- dreptunghiuri*, 372-374  
*elipse*, 374-375  
*poligoane*, 375-377  
*sectoare*, 374-375  
*suprafete de coardă*, 374-375
- fundal, 367  
 hașurare, 365  
 modele, 367-369  
 pensule cu culorile sistem, 369-371  
 pensule de stoc, 369-371  
 pensule nule, 370  
 selectare, listing, 371  
 ștergere, 372
- perioada inertă dintre cadre**, 672
- permanentă (proprietățile controalelor ActiveX)**, 624-626
- Persist (proiect SDI pentru serializare)**, 533
- personalizare**  
 bare cu instrumente, 292-299  
 bare de secționare, 464-465  
 bară de stare, 310-320  
 clase reprezentare, 464-465  
 controale ActiveX, 617, 619, 622  
 CRectTracker, clasă, 183  
 Print, casetă de dialog, 519
- PFM\_ALIGNMENT**, indicator, 453
- pictograme**, 46  
 comparație cu bitmap, 225  
 controale imagine, 231  
 creare, 223  
 documente (aplicații MDI), 482
- editarea pictogramei MFC implicite, 225
- evenimente generate de butoanele mouse-ului, 171-173
- importare, 229-230
- inserare, 226-227
- încărcare la momentul execuției, 231-236
- LoadStandardIcon(), pictograme furnizate de funcție, 172
- resurse, 226-227
- Picture Properties, casetă de dialog**, 231-232
- pixeli**, 512
- Play()**, funcție, 658
- PlayFreq()**, funcție, 662
- plăci de sunet**, 655-656
- plăci grafice**, 672
- POINT**, tip de date, 414
- pointeri**  
 g\_pDlgModeless, 220  
 pDX, 208  
 transmiterea între casete de dialog nemodale, 217
- poligoane**  
 desenare  
*penițe*, 364-365  
*pensule*, 375-377  
 mod de umplere, 375-376
- polimorfism (funcții virtuale)**, 508
- PolyBezier()**, funcție, 362-369
- PolyBezierTo()**, funcție, 364-365
- Polygon()**, funcție, 375-376
- Polyline()**, funcție, 364-365
- PolyLineTo()**, funcție, 364-365
- PolyTextOut()**, funcție, 379
- populare**  
 casete combinate, 121-124
- controale arbore, 125-128
- controale casetă cu listă, 128-130
- controale listă, 132-136
- PopulateCombo()**, funcție, 122
- PopulateListBox()**, funcție, 128
- PopulateTree()**, funcție, 125
- portret, orientare**, 523
- poștă electronică (MAPI)**  
 COM, 581-582  
 funcții, 674-677  
 MAPI extins, 675  
 SDK, 675  
 suport în infrastructură, 677-682
- PreTranslateMessage()**, funcție, 169
- previzualizare**  
 AppWizard, 504  
 suport, 505
- primar, buffer**, 657
- Print, casetă de dialog**, 519-522  
 accesare directă, 526-528  
 Collate, casetă de validare, 519  
 scurt-circuitare, 516
- PrintAll()**, funcție, 522
- PrintCollate()**, funcție, 522
- PrintRange()**, funcție, 522
- PrintSelection()**, funcție, 522
- privat, acces**, 428, 510
- procedură**  
 de fereastră implicită, 111  
 de întrerupere MCI, 685
- procese**, 651-652
- procesoare de text**, 449
- Process Viewer**, instrument, 652

- ProcessShellCommand()**, funcție, 256-257, 486
- programe**  
 guidgen.exe, 584-586  
 MDBView.exe, 675
- Progress Properties, casetă de dialog**, 138
- proiecte**  
 AppWizard, 23  
 Buttons, 72-74  
 CityBreak, casetă de dialog, 83-88  
 configurații  
*creare*, 52  
*stabilire*, 632  
 creare, 23  
 depanare  
*algoritmi de sortare*, 642  
*combinații de taste*, 640  
*indicatori de compilare*, 635  
*puncte de întrerupere, condiții*, 639  
 deschidere, 36  
 fișiere de asamblare, 51  
 inserarea controalelor ActiveX, 187  
 închidere, 33  
 opțiuni de compilare, 51  
 reprezentări, accesare, 38  
 resurse, 45  
 salvare, 33  
 SDICoin, 247  
 sub-proiecte, 38
- Project Settings, casetă de dialog**, 51, 632
- Project Wizard. Vedeți AppWizard**
- Properties, comandă meniul View**, 226-228, 32
- oportunitate dimensiunilor**, 12-515
- proprietăți**  
 butoane de comandă, 73-74  
 casete de dialog, 204  
 controale ActiveX  
*ambientale*, 628-629  
*editor de resurse*, 191  
*interfață*, 617-627  
*mapare*, 622-624  
*pagină pentru proprietăți generice de culoare*, 619  
*permanență*, 624-626  
*proprietăți generice*, 617-618  
*proprietăți specifice*, 619, 622  
*testare*, 628  
*tipuri*, 617  
 Group, 83-88  
 proprietățile elementelor de meniu, 273  
 subclasarea controalelor, 621-622
- protejat la scriere, acces**, 429
- protejat, acces**, 510
- protejat, operator**, 218
- protejate, variabile membru**, 261-262
- PtInRect()**, funcție, 181-182
- public, acces**, 510
- publice, funcții membru**, 481
- punct senzitiv**, 180
- puncte de întrerupere**  
 activare/dezactivare, 645  
 configurare, 644  
 depanarea proiectelor, 639
- puncte, desenare**, 358-360
- Push Button Properties, casetă de dialog**, 56-57, 73

## Q

- qsort()**, funcție, 642
- QueryInterface()**, funcție, 584, 589, 666
- QuickWatch, fereastră (depanator)**, 649

## R

- ramificație, noduri**, 116
- rand()**, funcție, 663
- rapoarte (aplicații de baze de date)**  
 actualizarea valorilor, 575  
 căutări, 572-575  
 schimb de câmpuri, 576
- rădăcină, nod**, 116
- .rc, extensie de fișier**, 50
- RDBMS (sisteme de gestionare a bazelor de date relaționale). Vedeți relaționale, baze de date**
- Read()**, funcție, 548-549
- ReadMail()**, funcție, 680
- RecalcLayout()**, funcție, 477-480, 481
- recent utilizate, lista fișierelor**, 545-546
- Recent Workspaces, listă**, 36
- RECT, structură**, 406
- Rectangle()**, funcție, 372
- redenumirea fișierelor**, 555
- redesenare, context dispozitiv**, 330, 332-335
- redesenarea barelor de dialog**, 309
- redesenarea ferestrelor**, 330
- redimensionarea ferestrelor**  
 dimensionare, eveniment, 405, 407-408

fixarea dimensiunilor, eveniment, 408-409, 412

formular, tip de reprezentare, 404

limitarea dimensiunilor, 414-415

redimensionare, casete de dialog, 415

**RedrawWindow()**, funcție, 328-330

reflectarea mesajelor de notificare, 446

regenerarea fișierului **ClassWizard**, 50

**RegisterClipboardFormat()**, funcție, 556-558

**Registers**, fereastră (depanator), 649

registru (automatizare **OLE**), 592, 600

relaționale, baze de date aplicații

- actualizarea valorilor*, 575
- conectare*, 570-572
- interogări SQL*, 572-575
- mulțimi de înregistrări*, 569-570
- schimbul de câmpuri*, 576
- suport (AppWizard)*, 568-570

definiție, 564

**ODBC** (conectivitate deschisă pentru bazele de date)

- aplicații*, 564-565, 569
- clase DAO*, 571
- configurarea surselor de date*, 566-568
- SQL*, 565-566
- schemă*, 564-565

**SQL** (limbaj structurat de interogare)

- definiție*, 565-566
- interogări*, 572-575

**Release()**, funcție, 583, 657, 672

**ReleaseCapture()**, funcție, 179, 613

**ReleaseDC()**, funcție, 667-668

**ReleaseDispatch()**, funcție, 601

**RemoveAll()**, funcție, 131

**RemoveView()**, funcție, 470

repetare

- buffere*, 658
- culori*, 667

**ReplaceSel()**, funcție, 107

**Repositioned**, eveniment, 615

reprezentare, clase, 464-465

reprezentări. *Vedeți și API; interfețe; ferestre*

- actualizare*, 267-268
- adăugare/înlăturare*, 470
- arbore, tip de reprezentare*, 441
- AppWizard*, 441
- control arbore*, 469
- editare imediată*, 447-449
- inserarea elementelor*, 442, 445
- nod selectat*, 446-448
- stiluri*, 442

bare de controale, 300

browser **HTML**, tip de reprezentare, 454-455

documente

- afișarea datelor*, 262-264
- relații*, 490

editare formatată, tip de reprezentare, 450

- formatarea paragrafelor*, 451, 453
- încărcarea/salvarea textului*, 451
- obiecte OLE*, 453

editor, tip de reprezentare, 450

înregistrare, tip de reprezentare

- controale de editare*, 578-579
- creare*, 576-577
- editare*, 577-578

listă, control, 119-121, 132-136

listă, tip de reprezentare

- coloane și antete*, 434, 437-438
- creare*, 427
- determinarea selecției*, 438-440
- inserarea elementelor*, 427-430
- stiluri*, 432-434

**MDI**, aplicații, 481-483

- accesare*, 489
- accesarea datelor*, 492-493
- actualizare*, 493-496
- clase*, 496-497
- creare*, 486-489
- inițializare*, 499-500
- meniuri*, 497-498
- reprezentare activă*, 481
- șiruri*, 498-499

proiecte

- accesare*, 38
- ClassView*, 38-44
- FileView*, 48-49
- ResourceView*, 45-48

reprezentări dinamice, 470-475

**Requery()**, funcție, 574

**ResetContent()**, funcție, 129

**Resource Symbols**, casetă de dialog, 47

**Resource Symbols**, comandă (meniul **View**), 56-57

**Resource**, comandă (meniul **Insert**), 226-228

**ResourceView** (fereastra spațiului de lucru al proiectului)

- aplicații SDI*, 253
- butoane de pe bara cu instrumente*, 297
- resurse*, 45-48

**RestoreDisplayMode()**, funcție, 672

resurse, 26

- adăugare*, 47
- aplicații MDI*, 481-483
- bare cu instrumente*, 47
- butoane*, 297-299
- inserare*, 300

bitmap-uri, 46

cod, separare, 45

controale **ActiveX**, 191

- proprietăți*, 191-192
- testare*, 613-614

cursoare, 46

editare, 48

fișiere, 56-57

localizare, 47

resurse dialog, 46, 304

resurse meniu, 46

- adăugare*, 270-271
- creare*, 270
- elemente de meniu*, 271-272
- identificatori de comandă*, 273
- titluri*, 271

supradefinirea funcțiilor, 48

tabele de acceleratori, 45

tabele de șiruri, 46-47, 313

versiune, 47

vizualizare, 45

rețele, depanare, 639

**RFX** (schimb de date între câmpuri și înregistrări), 576-577

ridicat, stare, butoane grafice, 235-237

**RoundRect()**, funcție, 372

**RPC** (apel de procedură la distanță), 587

rularea în fundal (**Developer Studio**), 592

**RUNTIME\_CLASS**, macrodefiniție, 254-255, 484

## S

salvare

- proiecte*, 33
- proprietățile controalelor ActiveX*, 624-626

**SaveModified**, funcție, 257-258

scalare liberă, moduri de mapare, 340-342

scheme, 534

schimb dinamic de date (**DDE**), 558

schimbul de date între înregistrări și câmpuri (**RFX**), 576-577

scrierea fișierelor, 548-551

**Scrollbar Properties**, casetă de dialog, 142

**SCROLLINFO**, structură, 145

**SDI**, aplicații (interfață document singular), 257

- adăugarea opțiunilor de meniu*, 265-266

bare cu instrumente, 253

bară de stare, 253

clase, 250-251

creare, 246-249

elementele ferestrei, 251-253

infrastructură

- funcții*, 255-260
- manipularea fișierelor*, 532-534
- reprezentare de tip arbore*, 441
- reprezentare de tip listă*, 427
- serializare*, 532-534
- tipărire*, 504

machete de document, 253-260, 265-266

reprezentare de tip formular, 404

**ResourceView**, pagină, 253

**SDICoin**, proiect, 247

**SDK** (kit pentru dezvoltarea aplicațiilor)

- Game SDK*, 655-656
- MAPI*, 675
- OLE DB SDK*, 655
- viteză*, 655-656

sectoare, desenare, 374-375

secundar, buffer

- creare*, 657-658
- mixare*, 658-659, 662-664
- redare*, 658
- repetare*, 658
- zgomot alb*, 657

securitate, controale **ActiveX**, 605-606

secțiunea spațiului de lucru al proiectului

- ClassView*, 38-44, 380
- Add Member Variable*, opțiune de meniu, 261-262

efectuarea de dublu clic asupra elementelor, 39  
 inserarea de variabile și funcții membru, 80  
 meniuri de context, 40-43  
 pictograme, 39  
 FileView, 48-49  
 pagini, 38  
 ResourceView  
   aplicații SDI, 253  
   butoane de pe bara cu instrumente, 297  
   resurse, 45-48  
**secțiuni, 315-321**  
**Seek(), funcție, 553**  
**Select Database Tables, casetă de dialog, 569-570**  
**Select Database, casetă de dialog, 568**  
**selectare**  
   penițe, 352-353  
   pensule, 371  
**SelectObject(), funcție, 352, 371**  
**selector de dată/oră, control, 154**  
   adăugare în casetele de dialog, 155-156  
   determinarea valorilor, 160  
   formatarea afișării, 158  
   formate de dată/oră, 157-159  
   inițializare, 157-160  
   inițializarea datei/orei, 159  
   interval, 157  
   manipularea obiectelor, 167  
   mesaje de notificare, 160-162  
   proprietățile calendarului, 160  
   rata de derulare, 164-165  
   stiluri, 155-156  
   validitate, 157  
   variabile, 156  
**semnal de undă, buffer. Vedeți buffere**  
**separatori (bară de stare), 274, 312, 313-314**  
**serializabile, clase**  
   declarare, 534-536  
   implementare, 536-539  
**serializare, 532-534**  
   date  
     Clipboard, 558-559  
     documente, 545  
     extensii de fișiere, 532  
     obiecte, 533-536, 539, 543-545  
   infrastructură SDI, 533  
   filtre, 533  
   Persist, proiect, 533  
   șiruri pentru macheta de document, 532-533  
   WordPad, 603  
**Serialize(), funcție, 259, 479, 533, 543**  
**serve**  
   procese (contexte), 586-588  
   serve de automatizare  
     adăugarea de metode, 597-599  
     biblioteci de tipuri, 598  
     COleObjectFactory, 595-596  
     containere, 602-604  
     creare, 591-597  
     DISP\_FUNCTION, macrodefiniție, 597  
     documente active, 602  
     încapsulare, 602-604  
     limbaj de definiție a obiectelor, 593-596

nume, 591-592  
 parametri în linia de comandă, 595-596  
 registru, 592  
 rularea în fundal, 592  
 tabela obiectelor aflate în rulare, 592  
 VB Scripting, 591-592  
 Word, 589  
 WordPad, 603  
**Set Active Configuration, comandă (meniul Build), 632**  
**Set Ambient Properties, comandă (meniul Edit), 628**  
**SetBitmap(), funcție, 231**  
**SetBkColor(), funcție, 382**  
**SetBkMode(), funcție, 383**  
**SetCapture(), funcție, 179, 612**  
**SetCheck(), funcție, 279**  
**SetClipper(), funcție, 667**  
**SetColor(), funcție, 164-165**  
**SetCommand(), funcție, 195**  
**SetCooperativeLevel(), funcție, 657**  
**SetCursor(), funcție, 231**  
**SetDateTimeSpan(), funcție, 167**  
**SetEnhMetaFile(), funcție, 231**  
**SetFirstDayOfWeek(), funcție, 164-165**  
**SetFormat(), funcție, 157**  
**SetIcon(), funcție, 231**  
**SetImageList(), funcție, 238**  
**SetLineSize(), funcție, 152**  
**SetMapMode(), funcție, 337-339**  
**SetModifiedFlag(), funcție, 258**

**SetMonthCalColor(), funcție, 160**  
**SetMonthDelta(), funcție, 164-165**  
**SetPageSize(), funcție, 152**  
**SetPalette(), funcție, 667**  
**SetPaneText(), funcție, 315**  
**SetPixel(), funcție, 327, 332**  
**SetPolyMode(), funcție, 375-376**  
**SetPos(), funcție, 141-142**  
**SetRange(), funcție, 165**  
   calendar, control, 164-165  
   indicator de evoluție, 140  
   selector de dată/oră, control, 157  
**SetRange32(), funcție, 140**  
**SetScrollInfo(), funcție, 145**  
**SetScrollPos(), funcție, 145**  
**SetScrollRange(), funcție, 144**  
**SetScrollSizes(), funcție, 417-418**  
**SetSel(), funcție, 107**  
**SetSelection(), funcție, 153**  
**SetStep(), funcție, 141-142**  
**SetSysString(), funcție, 591-592**  
**SetTabStops(), funcție, 112**  
**SetTextAlign(), funcție, 380**  
**SetTextColor(), funcție, 382**  
**SetTickFreq(), funcție, 152**  
**SetTimer(), funcție, 672**  
**SetToday(), funcție, 164-165**  
**SetViewportExt(), funcție, 341**  
**SetWindowExt(), funcție, 341**  
**SetWindowLong(), funcție, 434**

**SetWindowPlacement(), funcție, 404**  
**SetWindowPos(), funcție, 412**  
**SetWindowText(), funcție, 78, 681**  
**ShowControlBar(), funcție, 302**  
**ShowWindow(), funcție**  
   activarea/dezactivarea butoanelor, 78-79  
   afișarea/ascunderea butoanelor, 76-78  
**sistem**  
   casete de dialog modale, 220  
   culori de pensule, 369-371  
**sistem de operare, fișier, 547**  
**sisteme de gestionare a bazelor de date (DMS), 564**  
   moduri de mapare a paletelor, 347  
**SizeForm, exemplu, 407-408**  
**Slider Properties, casetă de dialog, 150**  
**software**  
   controlul versiunilor, 537  
   SDK (kituri pentru dezvoltarea aplicațiilor)  
     Game SDK, 655-656  
     MAPI, 675  
     OLE DB SDK, 655  
     viteză, 655-656  
**sortare**  
   algoritmi, 642  
   casete combinate, 115-116  
**Source Browser, 635-637**  
**spații de lucru, 24-25**  
   ClassView, pagină, 380  
   comparație cu proiecte, 38

**Split, comandă (meniul View), 458-459**  
**Spy++, instrument, 650-653**  
**SQL (limbaj structurat de interogare)**  
   definiție, 565-566  
   interogări  
   actualizarea valorilor, 575  
   aplicații de baze de date, 572-575  
   schimb de câmpuri, 576  
**Sqrt(), funcție, 594**  
**SRCINVERT, indicator, 336**  
**standard, bară cu instrumente, 292-299**  
   ancorare, 295-297  
   AppWizard, 292  
   creare, 293, 295  
   fereastră asociată, 294  
   indicator de stil pentru bară de controale, 294-295  
   OnCreate(), funcție, 293  
**stare (casete de validare), 89-92**  
**StartDoc(), funcție, 528**  
**StartPage(), funcție, 529-530**  
**statice, ferestre multi-secțiune**  
   creare, 464  
   crearea aplicațiilor pentru gestionarea fișierelor, 468-469  
   inițializare, 464-467  
**statice, funcții, 555**  
**StepIt(), funcție, 140-142**  
**stiluri**  
   calendar, control, 162-164  
   indicator de evoluție, control, 139

indicatori  
     *bare de stare*, 310, 315-321  
     *bare suport*, 319-321  
     *CRectTracker*, 182  
     listă, tip de reprezentare, 432-434  
 stoc, penițe, 351  
 stoc, pensule, 369-371  
 stocare  
     datele documentului, 261-262  
     proprietățile controalelor *ActiveX*, 617-627  
 Stop(), funcție, 658  
 strcmp(), funcție, 130  
 strictetea tipurilor, vectori, 539  
 sub-proiecte, 38  
 subclasare, 107-112  
 subclasare (controale *ActiveX*), 607  
     *CListBox*, *CTreeCtrl*, 117  
 controale  
     *clase ActiveX*, 609  
     *licențiere*, 628  
     *înregistrare*, 627  
     *math.h*, fișier de antet, 611  
     *OLE\_COLOR*, tip de date, 618  
     proprietăți, 617, 621-622  
     tratarea mesajelor, 607  
     funcții de tratare, 110, 607  
 subliniat, 388  
 submeniuri, elemente, 274  
 sunet. *Vedeți* audio suport  
     automatizare *OLE*, 590-529  
     *ODBC*, 569  
 suprafața de lucru, context dispozitiv, 328  
 suprafețe (*DirectDraw*), 667  
 suprafețe de coardă, 374-375  
 sursă, fișiere, controale *ActiveX*, 627  
 Sysinfo, casetă de dialog, 95-97  
 szBuffer, parametru, 124

**Ș**

șiruri  
     parametri, 124  
     tabele de șiruri, 46-47  
         aplicații *MDI*, 482  
         *ID\_INDICATOR\_MOUSE*, 313  
         resurse șir de caractere, 313  
     șiruri pentru machete de document, 532-533  
     terminator nul, 550-551  
 ștergere  
     butoane, 299  
     elemente de meniu, 290  
     fișiere, 50, 555  
     fonturi, 402  
     penițe, 353-354  
     pensule bitmap, 372-374

**T**

Tab Order, comandă (meniul *Layout*), 84-85  
 TabbedTextOut(), funcție, 379  
 tabela obiectelor aflate în rulare, 592  
 taste de acces  
     bare cu instrumente, 303  
     casete de dialog, 70-71  
     casete de editare, 95, 102  
     depanarea proiectelor, 640

dezactivare, 95  
 listă, 95  
 specificare pentru elemente de meniu, 274-275  
 taste mnemonice. *Vedeți* taste de acces  
 tăiat, 388  
 TECHNOLOGY, indicator, 343  
 test de clic, 180-181, 183-184  
 Test, comandă (meniul *Layout*), 84-85, 102, 614  
 testare  
     aplicații modificate, 32  
     casete de editare, 102  
     controale *ActiveX*, 613-614  
         *container de test pentru controale ActiveX*, 628-629  
         în editorul de dialoguri, 191  
         proprietăți, 628  
         proprietăți ambientale, 628-629  
     obiecte *OLE*, 603  
 text  
     afișare/fundal, culori, 382  
     alinie, 380-382  
         determinarea opțiunilor, 380  
         *TA\_UPDATECP*, indicator, 384-385  
     bară de stare, 315-321  
     casete combinate, 124-125  
     casete de editare  
         adăugare, 100-102  
         controale, 95  
         determinarea textului, 102-104  
         distincția minuscule/majuscule, 100

inițializarea câmpurilor, 102-104  
 mesaje de notificare, 104-105, 107  
 moștenirea claselor, 107-112  
 multi-linie, 112  
 numerice, 100  
 parole, 100  
 taste mnemonice, 95, 102  
 testare, 102  
 tratarea mesajelor, 104-105  
 variabile, 95, 102-103  
 culoare, 382  
 desenare, funcții  
     afișarea de text simplă, 379  
     alinieră textului, 380-382  
     culori de afișare/fundal, 382  
     decupare la dreptunghiuri, 384-385  
     text opac/transparent, 383  
 desfășurare pe rânduri, 402  
 editare formatată, tip de reprezentare  
     formatarea paragrafelor, 451, 453  
     încărcare/salvare, 451  
     obiecte *OLE*, inserare, 453  
 elemente de meniu, etichete, 280  
 etichetă statică, control  
     adăugare, 95-97  
     atașarea de variabile, 97

casete de editare, 95  
 formatarea textului, 94  
 identificator, 96  
 operații la momentul execuției, 95-100  
 fonturi  
     aldine, 388  
     *CFont*, clasă, 386  
     *CreateFont()*, funcție, 387-392  
     *CreatePointFont()*, funcție, 386  
     cursive, 388  
     decupare, indicatori, 389  
     desfășurare pe rânduri, 402  
     dimensionare, opțiuni, 390  
     dimensiune, 387, 507  
     familie, opțiuni, 390  
     funcții de afișare a textului, 379  
     indicatori de grosime, 388  
     iterare, 393-398  
     înclinare, 388  
     multi-linie, text, 400  
     *OnDraw*, funcție, 507  
     orientare, 388  
     selectare, 393  
     subliniate, 388  
     ștergere, 402  
     tăiate, 388  
     text formatat, 400  
     *TrueType* (indicatorul *TMPF\_TRUETYPE*), 390  
 formatat, 400  
 multi-linie, 400  
*OnContextMenu()*, funcție, 283  
 opac/transparent, 383

*TextOut()*, funcție, 379, 382  
     poziția mouse-ului (bara de stare), 313  
 timp, evenimente, 169  
 tipărire  
     acces direct, 526-529  
     *AppWizard*, 504  
     contexte dispozitiv, 510-512, 525, 526  
     coordonate  
         conversie, 511  
         localizare, 510  
     copii, 520  
     *DevMode*, structură, 520  
     dimensiuni de pagină, 523  
     ecran/imprimantă, relație  
         ajustare, 511  
         culoare pe imprimante monocrome, 524  
         rezultate diferite, 516  
     funcții de acumulare, 528  
     grafică  
         desenarea pe imprimantă, 508  
         dreptunghi de încadrare, 511  
         imagini pe mai multe pagini, 515  
         obiecte *GDI*, 524  
         *OnDraw()*, funcție, 504-508  
         proporția dimensiunilor, 512-515  
     inițializare, 523, 529-530  
     *MDI*, infrastructură, 504  
     optimizare, 525  
     opțiuni, 515, 519  
     orientare, 515  
     pagini  
         imagini pe mai multe pagini, 515

intervale, 515  
 paginare, 515  
 pagini de început/sfârșit, 515-518  
 previzualizare (implementare MFC), 518  
 Print, casetă de dialog, scurt-circuitare, 516  
 SDI  
   adăugarea codului, 504-508  
   infrastructură, 504  
   stare, verificare, 528  
   suport, 505, 511  
   suspendare, 525-526  
   WYSIWYG, 508  
 titlu, bare suport (Internet Explorer), 321  
 titlu, bară (Developer Studio), 39  
 Tools, comenzile meniului  
   Customize, 223  
   MFC Tracer, 575  
 TRACE, macrodefiniții, 639-642  
   listing, 640-641  
   rezultate, 642  
 TrackPopupMenu(), funcție, 281  
   indicatori de aliniere, 282  
   parametri, 281  
 TrackRubberBand(), funcție, 182-184  
 TranslateColor(), funcție, 618  
 TRANSPARENT, indicator, 383  
 transparent, text, 383  
 tratare, funcții  
   bara cu instrumente, butoane, 299-300  
   clic asupra butoanelor, 79-82

comenzi  
   creare, 265-266  
   editare, 267-268  
   funcții de tratare pentru interfața cu utilizatorul, 277-280  
   funcții de tratare pentru meniuri, 276-278  
 fixarea dimensiunilor, eveniment, 407-408  
 funcții de tratare pentru meniuri de context, 281-282  
 OnClose(), funcție, 220  
 OnGetMinMaxInfo(), funcție, 413-414  
 OnHScroll(), funcție, 424  
 OnSize(), funcție, 405  
 OnStartModeless(), funcție, 215  
 subclase, 110  
 tratarea evenimentelor  
   adăugare prin WizardBar, 413-414  
   BN\_CLICKED, 80  
   casete de editare, 104-105  
   controale ActiveX, 196-199, 612-614  
   evenimente generate de butoanele mouse-ului, 169-174  
   evenimente generate de clic asupra butoanelor, 74, 169-173  
   GetNextItem(), funcție, 439  
   WM\_DESTROY, 328  
 tratarea comenzilor de meniu, 276-280  
   activare, 278  
   dezactivare, 278

funcții de tratare pentru interfața cu utilizatorul a comenzilor, 278  
 funcții de tratare, adăugare, 276-278  
 meniuri de context, 284  
 tratarea mesajelor. *Vedeți funcții de tratare*  
 tri-stare, casete de validare, 88  
 TrueType, fonturi, 390  
 TVN\_ENDLABELEDIT, indicator, 447

## U

undă audio, dispozitiv, 195  
 Unlock(), funcție, 658  
 UnlockWindowUpdate(), funcție, 407-408  
 UpdateAllViews(), funcție, 495  
 UpdateData(), funcție, 87, 99, 208, 552  
 URL (reprezentare de tip browser HTML), 454-455  
 utilitare. *Vedeți aplicații; AppWizard*

## V

validare  
   datele din casete de dialog, 213  
   funcții, 43, 212-213  
 ValidateAge(), funcție, 213  
 valori (indicatori)  
   bară cu instrumente, laturi de ancorare, 296  
   bară de dialog, aliniere, 306  
   Seek(), funcție, 553  
 variabile  
   acces, 510  
   arbore, control, 117

bară de derulare, control, 143-144  
 calendar, control, 164  
 casete combinate, 115-116  
 casete de dialog, 206-207, 233, 235  
 casetă cu listă, control, 119  
 clase, 41  
 control de editare, 410  
 conținut, verificare, 646-649  
 CPrintInfo, funcție, 508-509  
 denumire, 42  
 etichetă statică, control, 97  
 funcții, 41  
 glisor, control, 151-152  
 imagine, control, 233, 235  
 incrementare, 42-43  
 indicator de evoluție, control, 139-140  
 inițializare, 42  
 selector de dată/oră, control, 156  
 variabile membru  
   casete de dialog nemodale, 217  
   ClassView, 80  
   DDV, 210  
   dialog, clase, 206-207  
   dispecer, clase, 193  
   document, clase, 261-262  
   MDI, aplicații, 490-492  
   OnMouseMove(), funcție, 176  
   protejate, 261-262, 492  
 Variables, fereastră (depanator), 647-649  
 VARIANT, 197

varianți  
   parametri (funcții proxim), 189  
   tipuri de date, 590-529  
 VB Scripting, 539  
 VBX (controale ActiveX), 189  
 vectori  
   COBArray, 541  
   indicatori, 310  
   strictețea tipurilor, 539  
   varianți, 590-529  
 VERIFY, macrodefiniție, 642-644  
 versiune finală, configurație, 24-25, 632-633  
 versiuni  
   control, 537  
   interfețe COM, 588-589  
   resurse, 47  
 VGA.DLL, 323  
 video  
   Direct3D, 673-674  
   DirectDraw  
     accesare, 667  
     agregare, 666  
     comutarea suprafețelor, 667-668  
     crearea obiectelor, 666  
     dublu buffer, 667-668, 671-672  
     funcții GDI, 667  
     HAL, 665  
     HEL, 665  
     liste pentru decupare, 667  
     ModeX, 665  
     nivel de cooperare, 667  
     obiecte, 664-667  
     palette, 667  
     suprafețe, 667

DirectPlay, 674  
 MCI, 681  
   comenzi, 682-685  
   dispozitive, 683  
   fereastră MCI, 689-692  
   mesaje de notificare, 685-688  
   tipuri de dispozitive, 682  
 View, comenzile meniului  
   ClassWizard, 84-85, 464-465  
   Properties, 226-228, 232  
   Resource Symbols, 56-57  
   Split, 458-459  
 virtuale, coduri de taste, 112  
 virtuală, funcție, 508, 543  
   OnScroll(), funcție, 421-422  
   tabele, 582-583  
 virtuală, memorie, 99  
 virtuală, realitate  
   Direct3D, 673-674  
   DirectPlay, 674  
 Visual SourceSafe, 536  
 Visual Studio, 21  
 viteză  
   animație, 655-656  
   dublu clic, eveniment, 174  
 vtable (tabelă de funcții virtuale), 582-583

## W-X-Y

WaitForVerticalBlank(), funcție, 667-668  
 Watch, fereastră (depanator), 647-649  
 Width(), funcție, 359  
 Window, meniu, comanda Split, 458-459  
 Windows NT, nume de directoare, 195



- Windows, interfață, 26
- WizardBar, tratarea  
mesajelor, 413-414
- WM\_CLOSE, mesaj,  
220-221
- WM\_COMMAND, mesaj,  
276
- WM\_CONTEXTMENU,  
mesaj, 281
- WM\_DESTROY, mesaj,  
328
- WM\_ERASEBKGND,  
tratarea mesajului, 334
- WM\_GETMINMAXINFO,  
419
- WM\_HSCROLL, mesaj,  
424
- WM\_MOUSEMOVE,  
mesaj, 175
- WM\_PAINT, mesaj, 330
- WM\_SIZE, mesaj, 411
- WM\_SIZING, mesaj, 405
- WM\_VSCROLL, mesaj,  
424
- Word, server de  
automatizare, 589
- WordPad, obiecte OLE,  
603
- Write(), funcție, 550
- WYSIWYG, tipărire, 508
- X Pos, 59
- Y Pos, 59

(Casetele hașurate sunt nou apărute în versiunea 6.0)

